PAC.C: A UNIFIED CONTROL ARCHITECTURE FOR PACKET AND CIRCUIT NETWORK CONVERGENCE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Saurav Das

June 2012

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Nick McKeown, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Fouad Tobagi

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Guru Parulkar

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost Graduate Education

To my parents

Abstract

Service providers today face several challenges. By all accounts Internet traffic is growing at 40-50% per year, necessitating costly upgrades to carrier infrastructure. Yet carriers do not see a commensurate increase in revenue, nor do they see relative reductions in capital and operational expenditures (Capex and Opex).

Part of the problem is that service providers today separately own and operate *two distinct networks*: packet-switched IP/MPLS networks and circuit-switched TDM/WDM Transport networks. These networks are typically planned, designed and managed by separate divisions even within the same organization, leading to substantial management overhead, functionality/resource duplication, and increased Capex/Opex. This is clearly an expensive and inefficient way to run networks. There have been other attempts to unify the control and management of circuit and packet switched networks – essentially run *one converged network instead of two* – but none have taken hold.

In this thesis, we propose a simple way to unify both types of network using an emerging concept called Software Defined Networking (SDN). SDN advocates the separation of data and control planes in networks; where the data-plane can be abstracted and represented to external software-controllers running a Network Operating System (NetOS). All network control functions are implemented as applications on top of the NetOS. The applications make control decisions that manipulate an annotated-map of the network presented to them and kept consistent by the NetOS. In turn the NetOS translates the map-manipulations into data-plane reality by *programming* the data-plane switch *flow*-tables via a switch-API like OpenFlow. As circuits can readily be defined as flows, the basic idea is that a common-flow abstraction fits well with both packet and circuit switches; provides a common paradigm for control using a common-map abstraction; and makes it easy to control, jointly optimize, and insert new functionality into the network. We call our SDN based solution **pac.c** for **p**acket **a**nd **c**ircuit **.**network **c**onvergence.

We defined the common-flow abstraction as flow-tables that take the form of lookup-tables in packet switches and cross-connect tables in circuit switches. Together with a switch-API like OpenFlow, which we extended for circuit switches, it abstracts away layer and vendor specific hardware and interfaces, while providing a flexible forwarding plane for manipulation by a common control plane. The common-map abstraction was defined as one which provides full visibility into both packet and circuit switched networks, while abstracting away the complexity of state-dissemination from applications, allowing the latter to be implemented in a centralized manner.

We built several prototypes to demonstrate and verify our architectural constructs. Our complete pac.c prototype emulates an inter-city carrier network, with access packet-switches in three cities, interconnected by hybrid packet-optical switches in the backbone, all under OpenFlow/SDN control. With this prototype, we verified the simplicity and extensibility of our architectural solution, compared to current state-of-the-art industry practice. More importantly, we presented qualitative architectural insights into why our solution fares better; and gave reasons why our control solution can succeed where GMPLS - the only previous attempt at unified control over packets and circuits - failed. Finally, we identified and demonstrated several *new* networking capabilities enabled at the packet-circuit interface, and offered architectural solutions to a number of deployment challenges faced by any new control solution.

To demonstrate the benefits of reduced Total Cost of Ownership (TCO), we designed and analyzed today's IP networks and contrasted it with a converged packet-circuit network based on our control architecture. We found nearly 60% Capex savings and 40% Opex savings. More importantly the savings are insensitive to varying traffic matrices and grow as we dimension the network for increasing traffic demand. And finally, we introduced the map-abstraction in MPLS networks and demonstrated how *existing* packet services like traffic engineering can be replicated in an SDN based network, without the complexities of the IP/MPLS control plane. In doing so we drew parallels with SDN based control for packets and circuits.

To summarize, we have proposed, designed, analyzed and demonstrated a converged IP/MPLS/Optical network architecturally based on SDN. The common platform helps reduce expenditures, provides existing services, and helps carriers innovate by easing the introduction of new revenue-generating services that differentiate them from other carriers. Our work is in the early stages but with further development, if these ideas are adopted by service providers, its main impact would be that they can remain profitable as the Internet grows. As a result they would then have greater incentive to invest in their networks, which in-turn could benefit society immensely.

Acknowledgements

I am proud to call myself Nick McKeown's student. The last few years have been an awe-inspiring journey. I have been a part, but I have also sat back and watched, as an idea was taken through inception, grown through hard work, passion and commitment and then unleashed into the larger networking community to spawn off what might one day become an industry of its own. Through it all, Nick still found time to guide me; to help me understand what makes a good networking systems researcher; and to instill confidence in me to challenge the status-quo, question pre-conceived notions, and dare to "change the world". I am deeply grateful.

Likewise it is hard to overstate the impact that my co-adviser Guru Parulkar has had on my growth and progress as a PhD student and reseracher. It is Guru that jump-started this work (literally by convincing Ciena's CTO Stephen Alexander to work with me). It is Guru who has on innumerable occasions found clarity in my clouded thoughts and muddled ideas. And it is Guru who has given me direction and sustained my drive through these long years. It is an honor to be Guru's *shishya*.

I would also like to thank Fouad Tobagi for all the classes I have taken with him and his invaluable comments and feedback on my thesis.

I reserve special thanks for three of my colleagues, each of whom has been instrumental in my thesis progress. Vinesh Gudla helped me put together the world's first prototype for an OpenFlow controlled packet-optical network; Yiannis Yiakoumis, moved out of his comfort zone to work with me to build the complete pac.c prototype and emulated WAN testbed; and Ali Reza Sharafat helped me demonstrate for the first time an MPLS service like Traffic Engineering without the IP/MPLS control plane. Each of them worked with me at a crucial stage of my PhD and helped me make forward progress for which I cannot thank them enough.

There are also a number of people in the industry I would like to acknowledge. Firstly I would like to thank Shinji Yamashita for believing in me and my story long before others did. Shinji-san was instrumental in getting my work off the ground by providing the resources and support I needed from Fujitsu, Japan. Likewise I am deeply grateful to Lyndon Ong, Daniel Getachew, Preeti Singh, Joe Berthold and all my friends in Ciena and Ciena-India, for working with me, training me, schooling me and supporting my work. I am also much obliged to Hans-Martin Foisel, Andreas Gladisch, Fritz-Joachim Westphal, Michael Duser, Christoph Gerlach and all my friends at T-Systems/ Deutsche Telekom for the wonderful summer I spent in Berlin learning how Transport Networks worked. Finally I would like to thank Ori Gerstel from Cisco for inspiration, guidance and encouragement to think along these lines.

It has been a privilege to be a member of the McKeown Group. Every single group member has directly or indirectly had a positive impact on my work. This work would simply not have been possible without the individual and collective efforts of each group member – one only has to comb the references to realize the significance and magnitude of their contribution.

Two key figures require special mention – Boris Grek and Jacob Sun. Boris is the whole reason why I started pursuing a PhD in the first place. His command over all things he touched inspired me to try and become more like him. And simply put, this PhD would have never gotten off the ground without Jacob's support – both financial and emotional. I am forever indebted to them.

Finally I would like to thank my support system. I thank my loving parents and all family in India for their unwavering confidence in me. And my friends, especially Altamash, She-Hwa, Emel and Gordon for their calming influence. My wonderful wife Shachi, for it takes special strength and heart to be the spouse of a PhD student. Without her support I would not be writing this today, for which I thank her with all my love. And we are still standing here today, thanks in no small part to the two wonderful kids – Arya and Sana – we had along the way.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Large scale networks are expensive to operate and service providers are always looking for ways to reduce their capital and operational costs. One approach involves reducing the number of different types of networks they own. This can be accomplished by converging the different services the networks offer on to one network. For example, many service providers have eliminated specialized core-telephony networks and converged voice services with data services on to IP networks.

This thesis however, is about the convergence of networks running at two layers. Large service providers such as AT&T and Verizon support two infrastructures – a Layer 3 IP network and an underlying Layer 1/0 optical transport network. Today these networks are run separately; they are planned, designed and operated by different groups of people, even if they are in the same organization.

Partly this separation is because of the different heritage of the two networks. Transport networks – with their telecom heritage – tend to be tightly managed and over-engineered for extreme reliability and redundancy. Detailed and sophisticated management systems have always been integral to the design of transport networks. On the other hand, the Layer 3 networks have precious few management capabilities. The general approach is to configure - in a distributed fashion - each large router, with many locally created scripts and tools and let automated control mechanisms take over.

The network technologies are also quite different - IP networks are packet-switched, while transport networks are circuit-switched[†] - and there is a lack of common control mechanisms that support both technologies in a simple, unified way.

Whatever the reasons are, one thing is clear - operating two networks with two completely different mechanisms is clearly more expensive and inefficient than running one converged network with a unified control mechanism. There have been other attempts to unify the control and management of Layer 3 and Layer 1 networks – in particular, GMPLS – which is overly complicated, and seems unlikely to be adopted. Even if it was used, GMPLS tends to preserve rather than break down the traditional separation between the two networks.

In this thesis, we propose a simple way to converge both types of networks based on an emerging concept known as Software Defined Networking (SDN). We use SDN principles to define a *common-flow abstraction* that fits well with both types of network and provides a common paradigm for control; and a *common-map abstraction*, which makes it simpler and easier to insert new functionality into a converged packet-circuit network.

In this chapter, we first introduce the two wide-area network infrastructures and highlight their main differences. We then state the problem as one where we wish to simplify and unify the management of Layer 3 and Layer 1 networks, so that the network can be jointly optimized to provide the best service for customers. We discuss the state-of the-art and briefly touch on reasons why previous approaches have not worked. We then introduce our unified control architecture by giving details on the two abstractions they are based on and discussing their benefits. Finally we summarize our contributions, and outline the rest of this thesis.

[†] Note that the use of the term 'circuit' in this thesis does *not* imply low bandwidth (kbps) telephony-circuits. Circuits in the optical transport network range from several hundred Mbps to tens of Gbps. The use of the term 'circuit' simply implies guaranteed-bandwidth that is provisioned before it is used. This thesis interchangeably uses the terms 'circuit-switching' and 'optical-switching'. We consider optical switches in the transport network that have digital switching fabrics (eg. time-slot switching) as well as photonic switching fabrics (eg. wavelength-switching). We do *not* consider forms of optical-switching that are not circuit-switched (eg. optical packet and burst switching).

1.1 The Transport Network and the Internet

Wide area IP networks form the backbone of the Internet today. IP networks are packet-switched, i.e. packets are individually switched hop-by-hop from source to destination by IP routers. However, the packets are physically *transported* between the routers in an underlying nation/world-wide network of optical fibers and circuit switches (Fig. 1.1). Collectively this underlying network is known as the Transport Network. We take a closer look at the two networks in the following sections.

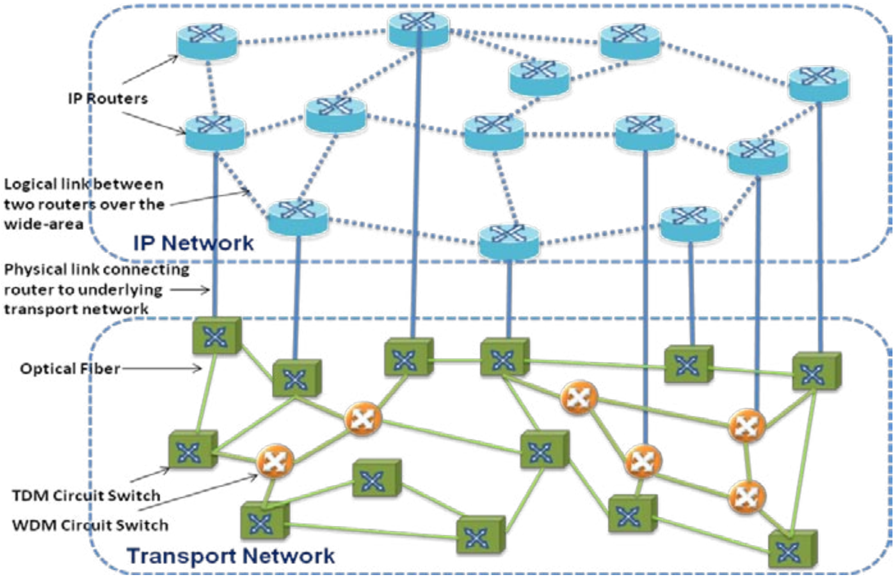

IP Routers
Logical link between two routers over the wide-area
IP Network
Physical link connecting router to underlying transport network
Optical Fiber
TDM Circuit Switch
WDM Circuit Switch
Transport Network

Figure 1.1: IP and Transport Networks

1.1.1 Internet Architecture

Architectural components of the Internet (layers, naming, addressing, protocols etc.) have been widely covered in several books. The Internet is a collection of interconnected IP networks. The constituent networks that make up the Internet have independent ownership, administration and management. To achieve global connectivity these networks use E-BGP to advertize IP address reachability and choose routes across routing domains known as Autonomous Systems (AS) [1].

This thesis is not about Internet architecture as a whole, but it does deal with the architecture of IP networks *within* an AS in the wide-area (WAN). A closer look at such intra-domain[†] IP core-networks reveal the following:

- IP networks have automated, fully-distributed control mechanisms. Such control mechanisms involve routing protocols (I-BGP, OSPF etc) and in some cases signaling protocols (LDP, RSVP etc) implemented in each router (Fig. 1.2). An IP router is both the control element which makes control decisions on traffic routing, as well as the forwarding element responsible for traffic forwarding. Control mechanisms are automated - after a router has been 'configured' (either manually or using scripts), the router automatically discovers its neighbors, the network topology, exchanges routing information, forwards packets, learns of failures and re-routes packets around them.

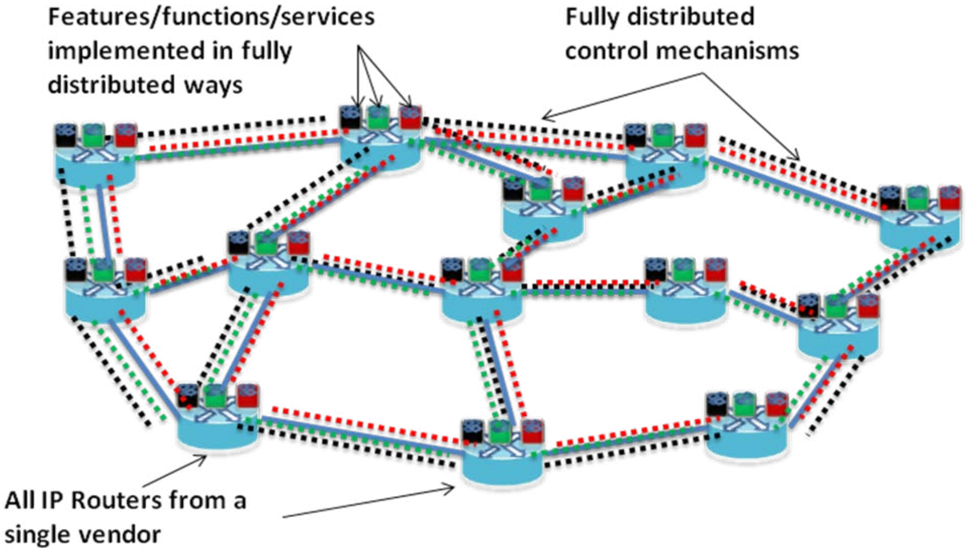

Features/functions/services implemented in fully distributed ways
Fully distributed control mechanisms
All IP Routers from a single vendor

Figure 1.2: Intra-domain IP Networks

- Services (or network functions/features) in IP networks, also tend to have fully-distributed implementations, which interact in subtle ways with the fully distributed control mechanisms (Fig. 1.2). These subtle interactions and the fully-distributed

[†] Intra-AS is sometimes also referred to as intra-domain in routing-protocol terminology. For example, OSPF, IS-IS and I-BGP are examples of intra-domain routing protocols while E-BGP is inter-domain. In this thesis we subscribe to the use of 'domain' from routing protocol terminology.

nature of their implementation make the features offered by an IP router-vendor non-standard (and as a result non-interoperable with other implementations), even though the control mechanisms are standardized. As an example, consider the network-function of traffic-engineering. Such a function today is provided by MPLS-TE (Traffic Engineering – discussed in more detail in Chapter 5). It *depends on* the IP/MPLS control plane which comprises of standardized protocols IS-IS and RSVP-TE. But the function of engineering traffic itself is proprietary and non-standardized. Traffic engineering on Cisco routers does not interwork with TE on Juniper or Huawei routers. And so, core IP networks are typically single-vendor networks.

- Networks perform poorly when congested. It has long been recognized that over-provisioning a packet network helps performance and end-user experience. Even though the public Internet remains best-effort end-to-end, Service Level Agreements (SLAs) and Quality-of-Service (QoS) guarantees exist *between* an IP network and its customers (such as other IP networks or large enterprises). Over-provisioning helps meet SLAs as well. And so we find that intra-domain IP core-networks are typically 2-4X over-provisioned.
- Management functions in IP networks involve configuration (typically via a Command Line Interface (CLI)), monitoring (typically via SNMP) and periodic maintenance. IP networks are generally perceived as hard to manage [2]. It is fair to say that IP management is ad-hoc and labor intensive. Teams of highly qualified personnel manually tweak the network, in the hope of achieving a balance between the local goals of each provider, and the global need to maintain connectivity.

To summarize, the Internet today provides a datagram, best effort service end-to-end; However the Internet is made up of intra-domain IP networks which are over-provisioned for acceptable performance and the need to meet SLAs; have automated, fully distributed control mechanisms; implement services in a distributed way making them typically single-vendor networks; and remain hard to manage.

1.1.2 Transport Network Architecture

The fundamental goal of a transport network is to provide communication bandwidth from one geographic location to another. For example, the IP link between two routers in a wide-area intra-domain IP network is a logical one – it may be established with a time-slotted circuit or with a wavelength-circuit in the transport network (Fig. 1.3). Here the IP network is regarded as a client to the transport network.

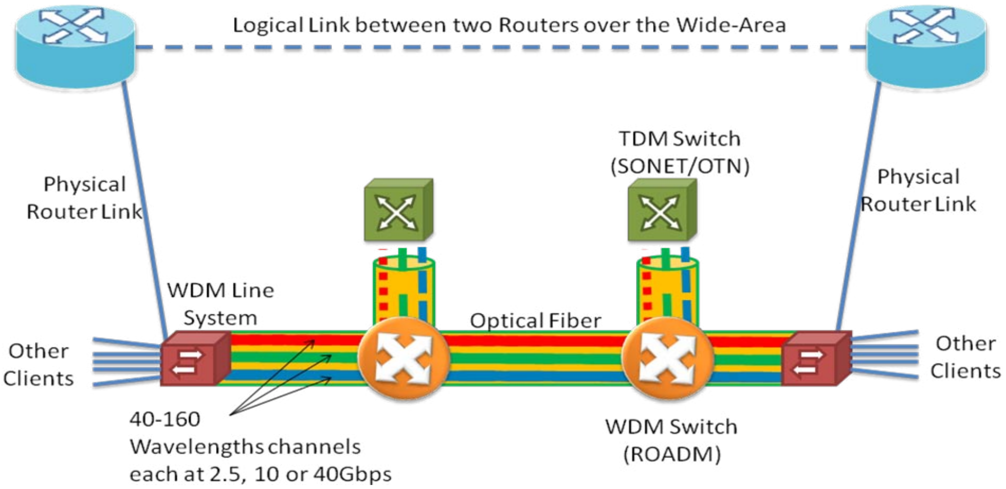


Figure 1.3: IP Router Connectivity in the Transport Network

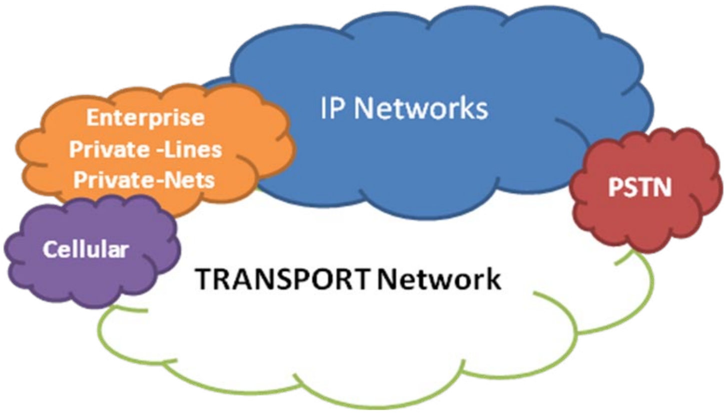


Figure 1.4: Transport Network and its Clients

Today transport networks support several “client networks”: IP core networks, the Public Switched Telephone Network (PSTN), the cellular network, point-to-point private-lines and enterprise private-networks (Fig. 1.4).

The transport network itself comprises of optical fibers with many (40-160) independent wavelength channels terminated at the WDM line-systems. The same wavelength in adjacent line systems may be stitched together to form a wavelength circuit (or path) via physical cables or via a wavelength-switch (WDM switch). Each wavelength channel operates at 2.5, 10 or 40 Gbps; 100Gbps wavelengths will be available in the near future. Because the wavelength channels operate at such high line-rates, the transport service provider often wants to sub-divide it to give sub-wavelength granularity connections to clients. Such granularity is provided by TDM switches.

The generic functional architecture of a transport network is described by the ITU in [3]. While we don’t go into deep details of the architecture and its terminology (which is substantial), from a high level it consists of several layers and partitions [4].

We make the following observations on transport networks:

- In contrast to IP networks, transport networks are always intra-domain (intra-AS), i.e. there is no equivalent to the Internet’s inter-domain (AS to AS) interaction. Instead the transport network describes *partitions* as “domains” (Fig. 1.5). Islands of equipment from different vendors, with different control or management procedures that *do not interoperate*, force the transport network to be partitioned into vendor-specific-islands[†]. So while the transport network is a multi-vendor network (unlike IP networks), they are not automated. In fact, transport networks are highly managed, where a hierarchy of Element and Network Management Systems (EMS/NMS) together with the OSS (Operations Support Systems) perform all control and management tasks (Fig. 1.5). In general, these systems are not programmatic interfaces, but full-blown GUIs which are vendor proprietary and triggered manually by teams of specialized network operators.

[†] With this definition, the control and management functions within an island are referred to as intra-domain and the interactions between islands as inter-domain. In this thesis, we will refrain from using “domain” to refer to vendor-islands to avoid confusion with the usage in IP networks. Also within a single vendor-island, there may be topological partitioning to improve the scalability of control procedures.

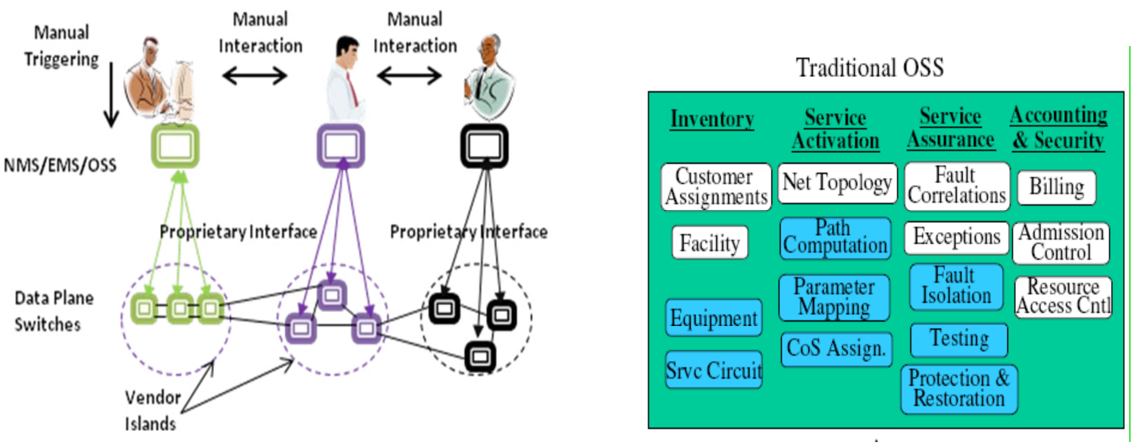


Figure 1.5: Transport Network Control & Management (OSS functions from [5])

- Providing a service in a transport network is a lengthy manual process. Consider a service supported by circuits in the data plane [4]: First, the path is planned by different planning groups in a piece-wise manner from one city to another. Then different provisioning groups execute the plan by manually configuring the transport network elements and their corresponding management systems along the path, and finally testing teams verify that the service is actually working and the associated management databases are being updated. It is easy to see why provisioning such a service takes weeks to months, and once up, why these circuits are *static* and stay in place for months or years.
- Traditionally, transport networks have lacked distributed control planes, but have always had a clean separation between data and management planes, where the EMS/NMS are physically separate from the data plane switches. They also tend to be more centralized in their control and management functions. Even when an automated distributed control plane exists, it does so within a vendor island making it proprietary and non-interoperable with other islands. Furthermore the automated control plane is still typically triggered manually via EMS/NMS.
- A transport network always provides hard guarantees in SLAs typically in terms of ‘big-pipes’ of bandwidth with high availability[†], delay and jitter bounds. For example

- guaranteed 10Gbps from point A to point B with 99.999% availability – the latter is known as five 9s availability which corresponds to about 5mins of downtime in a year. The 'big-pipe' granularity comes from the fact that in most cases, traffic has been aggregated the point where they require big pipes for transport. But it also stems from the fact that because it takes so long for a customer to 'get' such a service, the customer often prefers to get more at one time and keep it for a long time (static) without having to ask for more (and be subject to the long provisioning times).

1.2 Problem Statement

Service providers such as AT&T and Verizon today *separately own and operate* ***two*** *distinct wide-area networks*: packet-switched IP/MPLS networks and circuit-switched TDM/WDM transport networks. In fact, the biggest transport service providers (carriers/telcos) in the world are also the biggest Internet Service Providers (ISPs). For example, traditional carriers like AT&T, Verizon, British Telecom, Deutsche Telekom, NTT, Level 3/Global Crossing, Tata and others are also Tier 1 and Tier 2 ISPs [6].

These two networks are typically planned, designed and managed by separate divisions even within the same organization. Clearly owning and operating two separate networks is *inefficient*. At the very least, it leads to substantial management overhead from two teams of operators trained on different modes of operation and different management tools. But more importantly, it has a profound effect in terms of the Total Cost of Ownership (TCO).

Capex: To cope with Internet traffic growth (40-50% per year [7]), carriers would like to see lower Capex per Gbps[†] when upgrading their infrastructure. However, this has not been true in practice[*]. Operating two networks separately typically involves functionality and resource duplication across layers. Fault tolerance is a prime example: The underlying transport network often operates with 1:1 protection, while the IP

[†] For example, a 2X increase in cost for a 4X increase in capacity

[*] Upgrading from 10G to 40G links required more than 4X increase in equipment cost [9]

network running on top operates at less than 30% link utilization in preparation for unexpected traffic surges and link failures.

Opex: Operational expenditures can account for nearly 60-80% of the Total Cost of Ownership (TCO) of the network[^]. Such cost involve labor costs; costs for network Operations, Administration, Maintenance and Provisioning (OAM&P); equipment rack and PoP/CO[†] building rentals; and power consumption for operation and cooling. Separate operation of the two networks also involves time and labor-intensive manual coordination between the teams for service provisioning and maintenance tasks.

Service-Differentiation/Innovation: Service providers find it hard to differentiate their service-offerings from other carriers. Networks today are built using closed-systems (routers and switches) from the same set of vendors with the same set of features. The features are private “secret sauce” created inside each vendor's product. As a result, features are frozen inside each box, making innovation slow. The vendors have little incentive to innovate and create a barrier to entry for others, in both IP and transport networks.

Thus it is clear that from a service provider perspective, two separate networks that operate differently are inefficient. In networking, two is simply not better than one. In this thesis, we ask the question – *is there a way to run one network instead of two*? The problems outlined above led us to define three main goals underlying our work:

- To simplify and unify the control and management of IP and transport networks, so that the network can be jointly optimized to provide the best service for customers. Today, these planes are so different, and so complicated, that this is not feasible.
- Allow network operators to create and add new features to their packet and optical networks to provide revenue generating services that differentiate them from other carriers, thereby enabling a path of continuous innovation in the infrastructure.
- To allow network operators to use lower cost, lower power and more scalable optical Layer 1 transport switches in places they would use large, complex IP routers today.

[†] PoP – Point-of-Presence; CO- Central Office

[^] From private communications with several large carriers.

1.3 State of the Art

In this section, we discuss two topics that are state-of-the-art for IP and transport networks. The first involves a viewpoint popularly held by router-vendors. The second discusses the only previous attempt at unifying the control of the two networks.

1.3.1 IP over WDM

In this viewpoint, running one network instead of two can simply be achieved by *eliminating* circuit switching between the routers.

Recall that in Section 1.1.2, we stated that the transport network currently supports multiple client networks (Fig. 1.4). In recent years, there has been a trend to migrate the other client networks to the Internet. For example, traditional voice services are moving to IP, both at the end-user and in the service provider’s core.Meanwhile 4G cellular networks are also transitioning to all-IP networks for both data and voice. Previously (in 2G/3G) they used the IP network for data but circuit-switched networks for voice. Finally, point-to-point private-lines and enterprise private-network customers are increasingly moving to packet-network based solutions (eg. VPNs).

It is therefore entirely conceivable that in the near future, in contrast to Fig. 1.4, the *only* client for the transport network will be the Internet (Fig. 1.6(a)). In such a scenario, it is entirely valid to ask if there is a *need* for circuit *switching* in the underlying transport network.

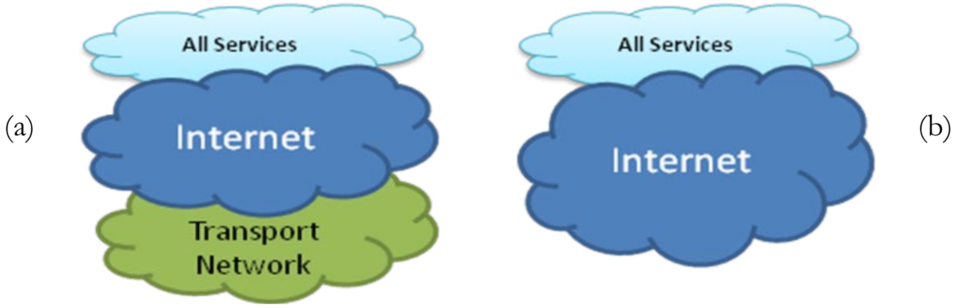


Figure 1.6: Possible Futures

For example, IP routers could be directly connected by point-to-point optical WDM links, in which case the WDM line systems are subsumed by the IP network (Fig. 1.6(b)) and transport switching is entirely eliminated – IP over WDM (no circuit switches).

We don't believe circuit switching will (or should) be eliminated. On the contrary, we believe that circuit switching is here to stay in the core, as it *can make the Internet more efficient*, with the caveat that for this to happen, the *two networks must work together dynamically*.

Fundamentally, packet switching is always going to be more expensive than circuit switching, simply because it performs a lot more functions, and does so at a much smaller granularity at much faster time-scales. In Appendix A, we show that our expectations are matched by numbers we obtain from real-world packet and circuit switches in the industry. Circuit switches are much more scalable; a circuit switch can switch much higher data rates, and consume much less power than an electronic packet switch. A useful rule of thumb is that an optical circuit switch consumes about 1/10th of the volume, 1/10th of the power and costs about 1/10th the price as an electronic packet switch with the same capacity (Appendix A). As a consequence, they are simpler, lower cost and more space efficient than an electronic packet switch.

This is not an attempt to say that packet and circuit switches are equivalent, because clearly they are not – while they both provide connectivity, they do so very differently. However, there are some functions that circuits can perform exceedingly well in the core – functions like recovery, bandwidth-on-demand, and providing guarantees (which we discuss in Chapter 3) – such that if circuits are eliminated, and those functions are then provided by packets, it comes at the cost of price (Capex), power consumption and size (Opex). In Chapter 4, we show the Capex and Opex inefficiencies of designing an IP-over-WDM network (without circuit switching), when compared to a packet network that interacts with a dynamic-circuit-switched network under common control.

On the other hand, a circuit switch doesn't have the statistical multiplexing benefits of a packet switch. This matters little at the core of the network where flows destined to the same next hop are naturally bundled, and their aggregate is relatively smooth [9]. Closer to the edge of the network, however, packet switching offers big benefits due to statistical multiplexing and more fine-grain control.

Thus we believe that packet switching is here to stay at the edge and dynamic-circuits offer significant advantages in the core. Indeed others have shown similar benefits [10, 79, 82-84].We do not know where the boundary between the edge and core lies, but preferably it is a flexible one. Diversity in the data plane is beneficial as both packets and circuits offer unique capabilities and cost vs. performance benefits in the data plane. But there is no real need for diversity in the control plane! And so the only way to *run one network instead of two* is to have a single converged control plane for packet and circuit switched networks. This thesis proposes a means for achieving such convergence.

1.3.2 MPLS/GMPLS

We are not the first to suggest a unified way to control packet and circuit switches. Most notably GMPLS offered an alternative approach [19], which has undergone standardization within the IETF (since 2000 [20]), and variations of the protocol suite have also gone through standardization at the ITU [21] and the OIF [22].

Generalized Multi-Protocol Label Switching (GMPLS) was designed as a superset of MPLS, and intended to offer an intelligent and automated *unified control plane* (UCP) for a variety of networking technologies – both packet and circuit. Its origin could be traced to the fact that MPLS already enforced a flow abstraction in core IP networks. Since circuits could readily be thought of as flows, a common-flow abstraction seemed natural. The logical next step involved developing a unified control framework on top of this common flow abstraction. And since MPLS already had a well developed control plane (derived from a well-developed IP control plane), GMPLS simply extended the same distributed routing and signaling protocols (OSPF-TE, RSVP-TE) to control circuit switches [5, 23-25].

However, despite a decade of standardization, implementation by transport equipment vendors and several interoperability demonstrations, GMPLS has yet to see even *one* significant commercial deployment as a *unified* control plane *across* packets and circuits. In fact, it isn't even used as a control plane *just* for transport networks [26, 27].

We do believe that the initial choice to use the concept of a flow in the data plane as the common abstraction was the right one. However, the subsequent choices either made or overlooked have contributed significantly to its failure. In the rest of this thesis, we will offer our perspective on where GMPLS went wrong, by highlighting these choices and comparing and contrasting our solution to them [29].

One fundamental observation we make here is that MPLS/GMPLS networks lack the common-map abstraction, and in principle all other deficiencies can be traced back to this observation. For example, without the common-map abstraction you lose the ability to implement control-functions in a centralized way. As a result features have to be implemented in a distributed way and be dependent in subtle ways on distributed protocols, increasing complexity and reducing extensibility (which we show in Chapter 3). Additionally using distributed protocols has its own issues with stability and being able to provide a gradual adoption path (Chapter 3). And without the common map you lose visibility across packets and circuits, which in turn makes services dependant on an interface such as the UNI [22], where the possible service requirements have been pre-supposed (pre-defined) and baked into the protocols, thereby hurting programmability and extensibility (also discussed in Chapter 3). Ultimately we argue that only control architectural changes will enable true converged operation.

1.4 Proposed Solution: Unified Control Architecture

Accomplishing the goal of a unified control plane for packet and circuit networks is not trivial. From our discussion in Sec. 1.1, it is easy to see that the control and management architectures of the two networks are vastly different (upper half of Fig. 1.7).

Furthermore the data plane units of packets and circuits (wavelengths, time-slots etc.) are also quite different from a control perspective.

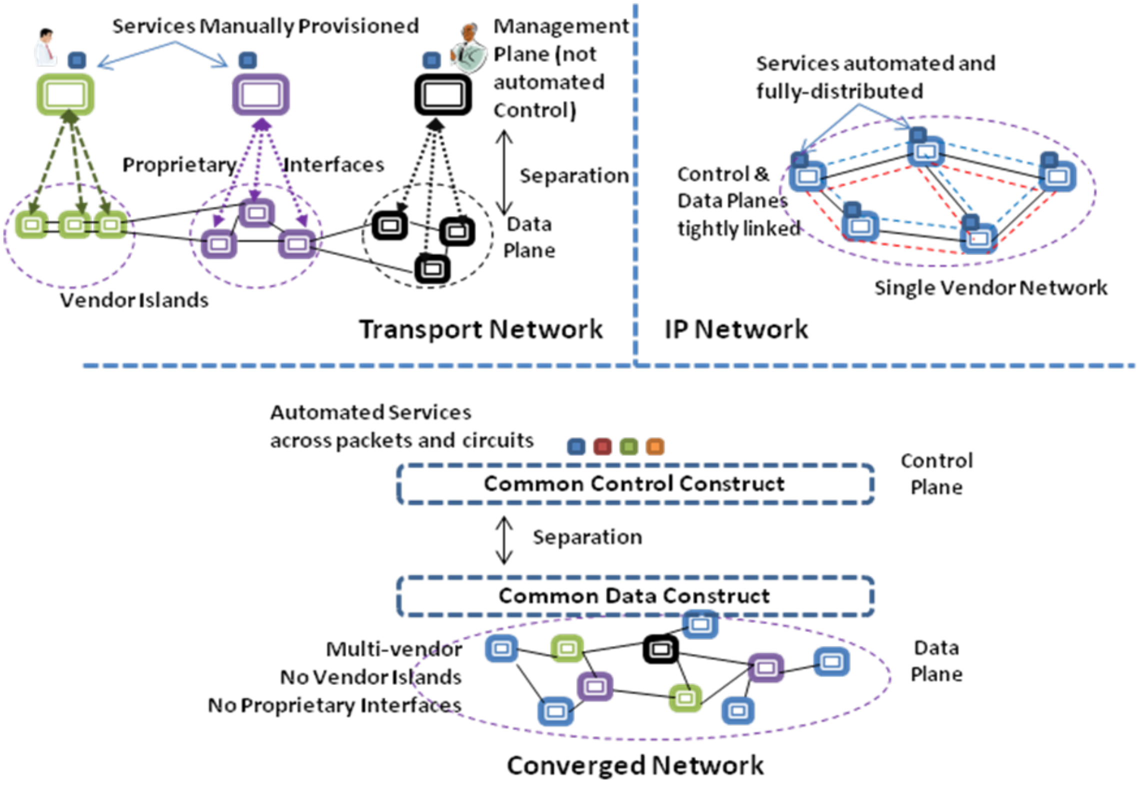


Figure 1.7: Path to Convergence

Thus in order to create commonly-operated converged packet and circuit networks, we ask ourselves the following questions (lower half of Fig. 1.7):

1. Can we find a common *data-plane construct* that would apply to both packets and circuits? Essentially a common data-plane abstraction that would provide a common paradigm for simple multi-layer control; one that allows flexible and vendor agnostic interfaces that can eliminate vendor islands and proprietary interfaces in running multi-vendor networks?
2. Can we develop a separate, common *control construct* that represents the networks in a common way? One that eases the development and fast deployment of automated

network-functions and services across packets and circuits, while giving the network operator the choice of selecting the best mix of technologies for their service needs?

We believe that such constructs are indeed possible, but require changes in control architecture. And so we propose a unified control architecture which has its underpinnings in two abstractions – the common flow abstraction and the common-map abstraction [11, 13]. Our work is heavily influenced by an emerging new direction in packet networks called “Software Defined Networking (SDN)” [16, 17]. SDN origins lie in fostering innovation in campus networks, by enabling researchers and network operators to experiment with new ideas ‘in’ the networks they use every day [12]. It was born out of related work that looked at security management issues in enterprise networks (ETHANE project [18]). We have applied SDN ideas to circuit-switching and carrier networks to propose a solution to converged operation of IP and transport networks.

1.4.1 Common Flow Abstraction

In a *traditional* IP backbone network, routers use a ‘datagram’ model, where they deal with each packet in isolation, rather than viewing the packets as part of some end-to-end communication. In other words, treatment given to a packet is independent of what packets came before or what packets might come afterwards.

But data-packets are naturally part of flows – i.e. they are naturally part of some communication (not necessarily end-to-end). Packets that are part of the same communication have the same *logical association*. Consider Table 1.1: it shows multiple different definitions of flows (i.e. different logical associations between packets); gives examples of what these flows can represent, and in the last column presents ways to identify these flows from fields in the packet-headers. The concept of flows is not a new one. In core networks today, flows exist as FECs coupled with LSPs (we discuss this in more detail in Chapter 5). To define the common-flow abstraction, we define flows in the following way:

Flow definition: logical association of packets	**Packet flow examples**	**Flow-identifiers from packet header fields**
End – to – End flows	All packets sent when we watch a YouTube video All packets in a file transfer All packets in a VoIP call (in one direction)	Well known 5-tuple of - IP source, IP destination, IP protocol (payload identifier) and Transport layer source and destination (TCP, UDP etc. ports)
Common-destination flow	All packets destined to China	IP destination prefix for China
Common-source flow	All packets from an enterprise branch location	IP source prefix for addresses allocated to enterprise branch
Common- traffic type flow	All web traffic	IP protocol (to identify TCP) and TCP dest. Port numbers to identify HTTP
Common- traffic type flow from a particular end-host	All voice traffic from a handset	MAC source (to identify end-host), Eth-type (to identify IP packets) and IP ToS (to identify traffic type)
Common - source and destination routers	All traffic between two routers	MPLS label id

Table 1.1: Packet flow definitions, examples and identifiers

Packet-flows: If a) packets are classified by their logical association and such soft-state is retained in switches to remember the nature of the flows which are passing through them; and b) the same set of actions are performed on all packets that have the same logical-association (flow-definition); and c) resource management and accounting of packets are done as part of flows; then the fundamental data-plane unit to control *is the flow* (and not the individual packets). Instead of the datagram, the 'flow' becomes the fundamental building block in packet networks.

Circuit-flows: In circuit networks, the fundamental data-plane unit to control *is the circuit*. And circuits are flows too – the circuit itself is the logical association for the data that is being carried in it between two end-points in a network. Only the flow identifiers for circuit flows are different from packet-flows. For example a circuit-flow could be:

- a time-slot on multiple links, making up a TDM signal-path, or
- a collection of time slots, on multiple different paths, bound together, or
- a single wavelength-path, or
- a wavelength path comprising different wavelengths along the path, or
- a set of timeslots on a particular wavelength, or
- a waveband – i.e. collection of wavelengths along a path.

Note the similarity of the above example for circuit-flows to the examples for packet flows in the middle column of Table 1.1.

Common-Flow Abstraction: It is easy to see that in most cases, the information identifying the logical association of packets in a packet-flow *exists* within the header fields of all the packets in the flow. And while the identifiers of circuit flows are different, both sets of identifiers can be placed in forwarding tables found in both packet and circuit switches (Fig. 1.8).

For packet switches the forwarding tables take the form of lookup-tables which can match incoming packet header-fields to ‘rules’ that define the flow. These rules are combinations of flow-identifiers (right-most column in Table 1.1) that embody the logical association between packets that are part of the same flow. Most packet-switches support lookup tables that match on only one kind of identifier – eg. Ethernet switches match on MAC addresses, IP routers match on IP addresses etc. But all packet-switches also support other tables (eg. Ternary CAMs) which allow flexible rule definitions that include combinations of identifiers as well as ‘don’t cares’. These tables support flexible rules-definitions for packet flows, and perform the same actions on all packets that match the rule. Thus with a packet-switch abstraction of <matching-rule, actions, statistics>, an incoming <packet, port> can be translated to an outgoing <packet′, port′> [12].

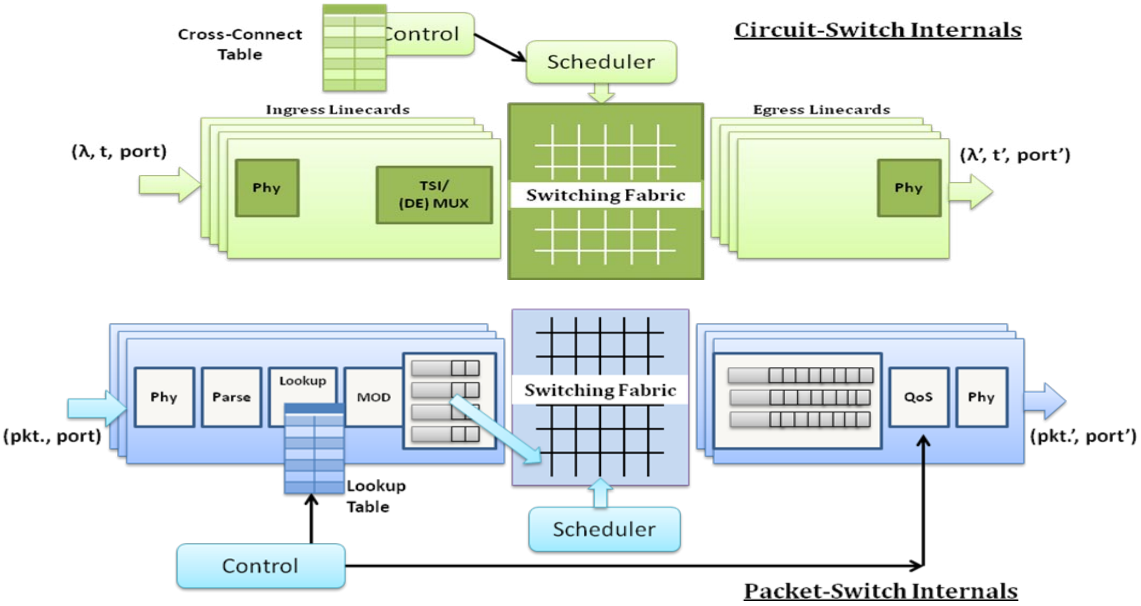


Figure 1.8: Packet Switch and Circuit Switch Internals (Appendix A)

For circuit switches the forwarding tables are cross-connect tables that control the scheduling of the circuit switch fabric to create a circuit within the switch. In circuit switches the forwarding table is not in the datapath. Nevertheless, the table supports a translation of an incoming wavelength, time-slot or fiber-port (λ, t, port) to an outgoing (λ′, t′, port′). Thus a circuit switch can also be abstracted as a table that supports <cross-connect rules, actions, statistics> [13].

The common-flow abstraction (Fig. 1.9) is therefore a common-forwarding abstraction (or a common-table abstraction), where we *abstract away all kinds of packet and circuit switch hardware*, by presenting them as forwarding tables for direct manipulation by a switch-API. In other words, switches are no longer viewed as Ethernet switches, IP routers, L4 switches, MPLS LSRs, SONET switches, OTN switches, ROADMS, or multi-layer switches – they are just tables; tables that support the flow identifiers *irrespective* of which traditional layer of networking (L0-L4), or combination of them, the flow may be defined with. In Appendix B we describe a switch-API for manipulating our common-table abstraction.

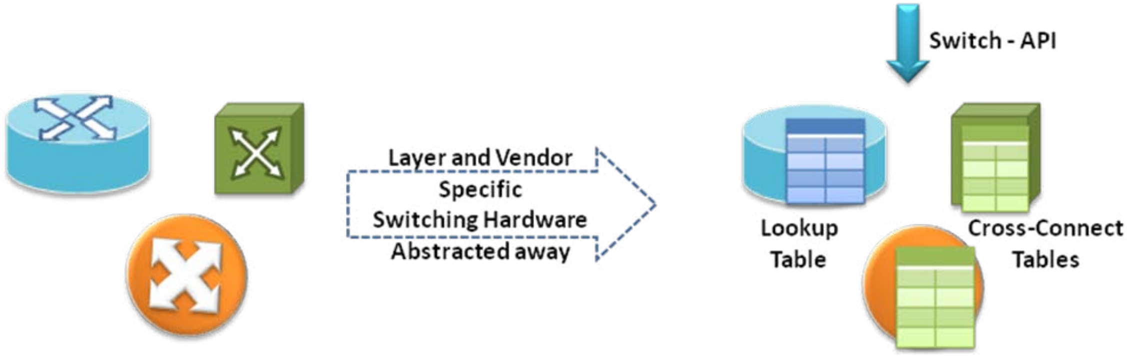

Switch - API
Layer and Vendor Specific Switching Hardware Abstracted away
Lookup Table
Cross-Connect Tables

Figure 1.9: Common-Flow Abstraction

Benefits of the Common-Flow Abstraction: The main benefits of the common-flow abstraction are:

- Simple, Flexible Multi-layer Control: Today's networks require multiple independent control planes for *switching* in different layers – for example Ethernet switching has its own set of control-protocols (STP, LACP, LLDP etc); IP has its own (OSPF, iBGP, PIM etc); so does MPLS (LDP, RSVP, MP-BGP); as well as SONET(RSVP-TE, OSPF-TE); OTN and ROADMs have proprietary solutions. The common-flow abstraction eliminates the need for multiple independent distributed control planes, by giving an external controller the ability to define packet and circuit flows flexibly and map them to each other, irrespective of which traditional layer of networking the flow identifier may belong to – from Layer 0 to Layer 4. In a way the common flow abstraction *de-layers* networks by treating packets and circuits as part of flows. The immense direct benefit is a reduction of complexity in control planes. In Chapter 3 we show an example of how we control flows on the basis of Layer 4 (TCP/UDP), 3 (IP), 2 (VLAN) and 1 (SONET) identifiers and quantify the reduction in complexity of our control plane implementation compared to industry solutions.
- Vendor-agnostic Control: The common-table abstraction together with the use of the switch-API makes our solution independent of vendor-specific solutions. Carriers benefit from this as packet-networks need no longer be single-vendor networks (Sec.

1.1.1) while remaining fully automated and feature-rich. Similarly carriers can eliminate multiple non-interoperable vendor-islands in transport networks (Sec. 1.1.2). The ability to run multi-vendor automated converged packet-circuit networks fully interoperable in the control plane provides an economic benefit which we will quantify in Chapter 4.

To summarize, the common-flow abstraction is a common-forwarding abstraction that lets us think of packets and circuits commonly as flows, thereby providing a common paradigm for flexible and vendor-agnostic control across multi-layer networks.

1.4.2 Common Map Abstraction

We find that in modern networks, there are several functions that we need from the network itself– examples of these functions are routing, access-control, mobility, traffic-engineering, guarantees, recovery, bandwidth-on-demand – the list goes on. Some of these may apply to packets-networks, some to circuits-networks, and some to both.

Ultimately these functions are implemented as control programs. And these control programs are easiest to write when they operate in a centralized way, with a global view of the network – both packet and circuit. Thus the second abstraction is a common-map abstraction across both packet and circuit switched networks. The global-map is an annotated graph of the network topology which we describe next.

Common-Map: The global-map is a database of the network topology (Fig. 1.10). It is a collection of network nodes - both packet and circuit switches. The node’s switching capabilities are represented by their forwarding tables (the common-flow abstraction) together with the features the tables support (match-fields, cross-connections, forwarding actions, mapping actions etc.). The switch information also includes collections of entities such as ports, queues, and outgoing link information. Example attributes of each entity are listed in Fig. 1.8. This database is created, presented to control applications and kept up-to-date by the map-abstraction.

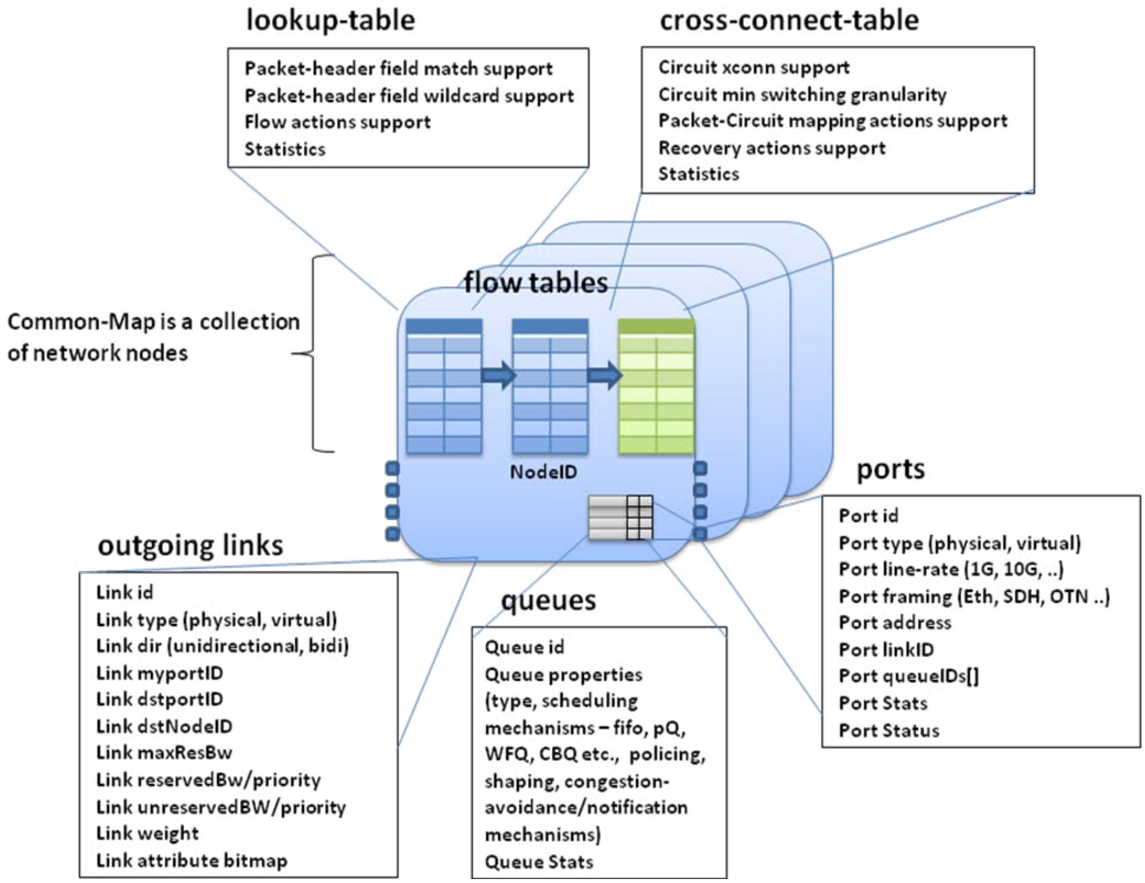

lookup-table
Packet-header field match support
Packet-header field wildcard support
Flow actions support
Statistics
cross-connect-table
Circuit xconn support
Circuit min switching granularity
Packet-Circuit mapping actions support
Recovery actions support
Statistics
Common-Map is a collection of network nodes
flow tables
NodeID
ports
Port id
Port type (physical, virtual)
Port line-rate (1G, 10G, ..)
Port framing (Eth, SDH, OTN ..)
Port address
Port linkID
Port queueIDs[]
Port Stats
Port Status
outgoing links
Link id
Link type (physical, virtual)
Link dir (unidirectional, bidi)
Link myportID
Link dstportID
Link dstNodeID
Link maxResBw
Link reservedBw/priority
Link unreservedBW/priority
Link weight
Link attribute bitmap
queues
Queue id
Queue properties (type, scheduling mechanisms – fifo, pQ, WFQ, CBQ etc., policing, shaping, congestion-avoidance/notification mechanisms)
Queue Stats

Figure 1.10: Annotated Graph (Database) of Network Topology

Aside from the nodes and links another database for flow-state can be created if the control applications need to retain such information. Fig. 1.11 shows the entities involved in retaining packet and circuit flow-state and their attributes. While the flow (both packet and circuit flows) databases can be a part of the common-map as they reflect network state (Fig.1.9), the decision to retain such state is left up to the control application. This is because flow-state is typically ephemeral, and knowledge of individual flow-state is usually not necessary for making control decisions. What is necessary is the *aggregate* effect the flows have on various parts of common-map. Such aggregates are reflected in the statistics maintained in common-map entities (tables, ports, queues etc.).

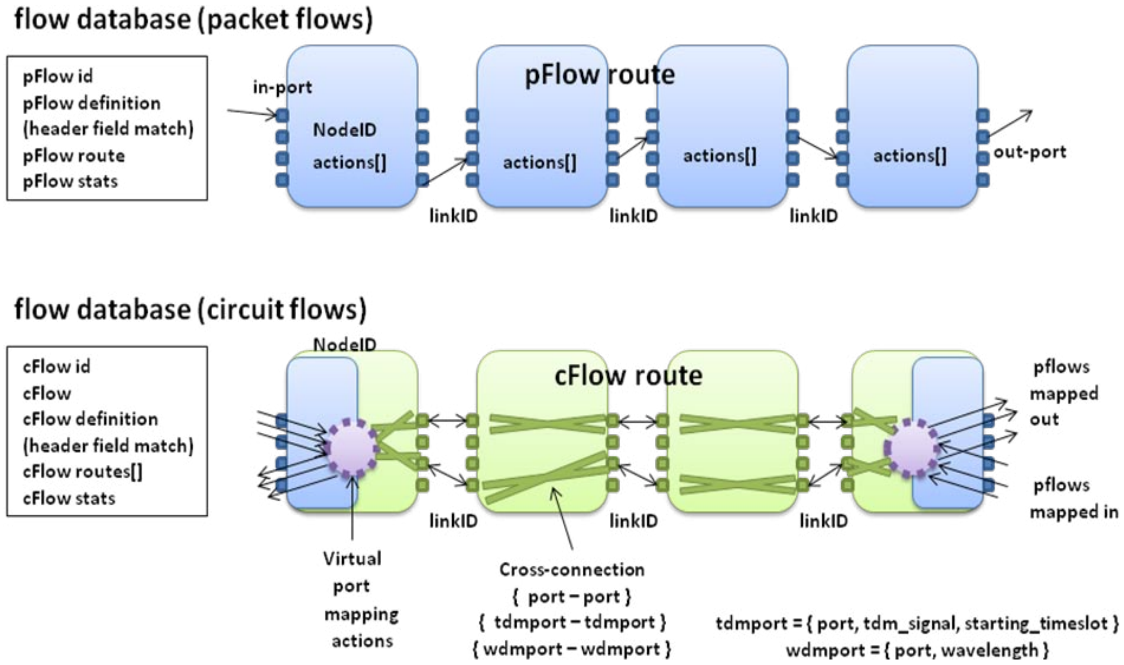


Figure 1.11: Flow Databases

Common-Map Abstraction: The global map and databases shown above are at the heart of the common-map abstraction. It allows control programs to be implemented in a centralized way, where they could be implemented to take advantage of both packets and circuits, using a *network-API* which manipulates a global-view of the networks. In Appendix C, we describe a network-API for applications across packet and circuits.

The common-map abstraction abstracts away the following (Fig. 1.12):

- The control program need not worry about how the map is being created and how it is being kept up-to-date. All it knows is that it has the map as input; it performs its control function; and delivers a decision.
- The control program also does not need to care about how that decision is compiled into forwarding plane identifiers and actions (using the common-flow abstraction), and then distributed to the data plane switches. State collection and dissemination have been abstracted away from the control program by the common map abstraction.

- Finally, each individual control function does not have to worry about conflicts that may arise between decisions it makes and decisions made by other control functions. Thus application-isolation is part of the abstraction provided to control functions.

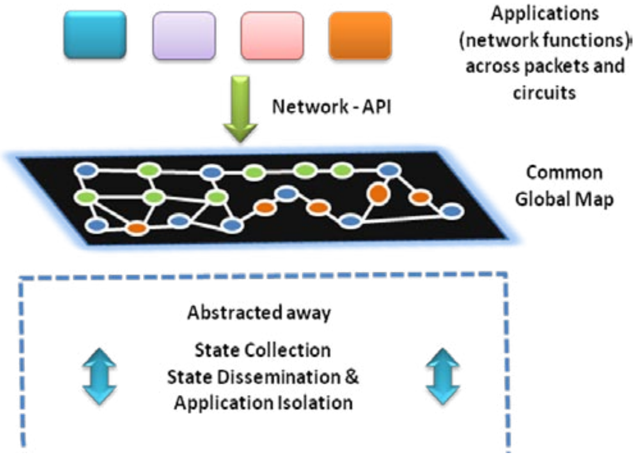


Figure 1.12: Common-Map Abstraction

Benefits of the Common-Map Abstraction: The main benefits of the common-map abstraction are:

- Programmability: Instead of defining network behavior up-front and baking it into the infrastructure, the common-map abstraction helps networks become programmable. It eases the path to innovation by offering a network API to programs for controlling network behavior. What does the network API include? Today, the three networking tasks of: i) configuring switches; ii) controlling forwarding behavior; and iii) monitoring network state; are performed separately. Configuration typically uses a CLI or NMS/EMS, forwarding state is determined by distributed routing/signaling or other special purpose protocols, and monitoring is done via SNMP, NMS/EMS, Netflow, sFlow etc. The network API can present calls for all three tasks together to network applications.
- Simplicity & Extensibility: The common-map abstraction breaks the chains that bind together today's distributed-implementation of network services to the state-

distribution mechanisms that support them. With the common-map abstraction the distribution mechanisms are abstracted away, so the *control function* can be implemented in a centralized way. Centralization makes implementing individual control functions simpler; but just as importantly the abstraction makes inserting *new* control functions into the network easy (extensible). This is because the state-dissemination problem has been solved *once* and abstracted away, so new control-programs do not have to worry about it by creating new distribution mechanisms or changing existing ones. We will show examples of simplicity and extensibility in writing control-programs in Chapter 3.

- Joint & Global Optimization: The common map-abstraction offers full visibility across packets and circuits. In other words it offers applications the ability to perform joint-optimization of network functions and services across both technologies; leveraging off the different-benefits of both packet and circuit switching; and doing so with a global view of the network.
- Choice: With the common-map abstraction and its global view, *new features* can be supported that take advantage of both packets and circuits. Additionally it allows the network programmer the choice of writing control programs in a variety of ways in the context of packet and circuit networks. A particular control program could still treat the packet and circuit flows as if they were in different *layers*, where they would have separate topologies, but still be commonly controlled. A different control program could treat them as part of the same layer with a single topology (and still commonly controlled). Yet another control program could go further and treat them as separate topologies while completely ignoring one of them. The common-map abstraction does not preclude any of the cases, and we will give examples of all of these cases in Chapters 3 and 5. In other words, with the common-map abstraction, the control function programmer/network operator has maximum flexibility to choose the correct mix of technologies for the services they provide.

1.4.3 Unified Control Architecture

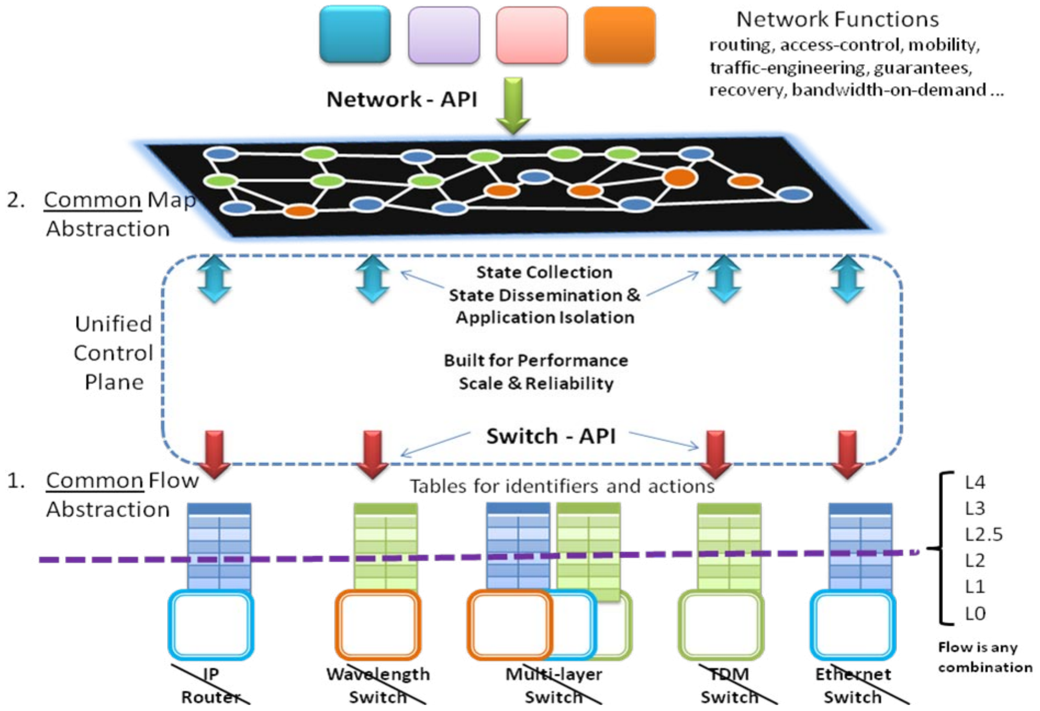


Figure 1.13: Unified Control Architecture

To summarize, our control architecture has its underpinnings in the two abstractions we have discussed in the previous two sections. If we combine Fig. 1.9 & Fig. 1.12, we can develop a more complete picture of the unified control architecture in Fig. 1.13.

The job of the unified control plane that sits between the common-map and the switches is three-fold. First, it provides the functions of state collection/dissemination and conflict resolution, which were abstracted away from the control functions by the common-map. Second, it includes the interface that instantiates the common-flow abstraction by providing the switch-API. Third the control plane should be engineered so it can scale to large carrier networks without suffering from poor performance and reliability. In the next chapter, we discuss how we have instantiated the common-flow and common-map abstractions using an interface called OpenFlow [14] and a Controller (external to the data-plane switches) running a Network Operating System (NOX [15]).

1.5 Contributions of Thesis

This thesis makes the following contributions:

Architecture: We have proposed and defined a unified control architecture for the converged operation of packet and circuit switched networks. The architectural underpinnings of our proposal include:

- A common-flow abstraction: that provides a common paradigm for flexible control across packet and circuit switches. We instantiated the common-flow abstraction by first creating a flow-table based abstraction for different kinds of circuit switches. We took into account switching-fabric types and port/ bandwidth representations, as well as various ways in which packet and circuit switches can be interconnected based on interface-type, framing method and line-rates (Ch.2). We also developed a switch-API for creating, modifying and deleting circuit flows; mapping packet-flows to circuit-flows and back with suitable adaptations; neighbor discovery and recovery messages; and finally error and statistics messages for circuit ports, links and flows (Ch.2 and Appendix B). OpenFlow v1.0 [28] was extended to include this API.
- A common-map abstraction: that liberates network control functions from the task of distributed state collection and dissemination. We extended an existing Controller called NOX [15] to simultaneously control both packet and circuit switches, thereby creating an instance of the common-map abstraction. We also developed link-discovery methods that do not preclude a layering choice and created a network-API for applications to manipulate the common-map (Ch.2 and Appendix C).

Validation of Architectural Benefits: We implemented our architectural approach in several prototypes (named **pac.c** for **p**acket **a**nd **c**ircuit **.c**onvergence) to validate the simplicity and extensibility of our approach:

- We built three pac.c prototypes - the first two systems demonstrated common control over packet switches and two different kinds of circuit switches – a TDM based one and the other a WDM based one. The more complete pac.c prototype, was used to emulate an inter-city wide-area network structure, with packet switches in 3 cities interconnected by circuit switches in the backbone, all under unified operation.

- We validated the simplicity of our proposed solution by:
    - Implementing and demonstrating a network-application across packets and circuits on top of our prototype emulated WAN – the network-application's goal was to treat different kinds of network-traffic differently.
    - Comparing our work to existing network control solutions - we found that implementation of the defined control-function in our control architecture takes 2 orders of magnitude less lines-of-code compared to existing solutions.
- We validated the extensibility of network-control functions in our architecture by:
    - Identifying and demonstrating multiple networking-applications across packets and circuits on our pac.c prototype. Examples include: Variable Bandwidth Packet Links; Dynamic Optical Bypass; Unified Routing & Recovery. The applications suggested are by no means the only possible applications, as service providers can define their own applications to meet their service needs.
    - Comparing our work to existing network control solutions – we show how existing rigid control interfaces cannot reproduce our network applications exactly, nor can they easily add new services given the tight coupling between applications and state distribution mechanisms.

Design & Analysis: We designed WAN infrastructures and performed Capex and Opex analyses on them to validate cost-savings from operating a network with both packet and circuit switching if done from a single control viewpoint -- i.e. using our unified control architecture.

- We outlined a design procedure for IP over WDM networks (reference design) and applied a cost-model to the components. Our Capex analysis for this reference design is more detailed than previous attempts, as we include access routers and dimension the IP network for recovery and traffic uncertainty. We accounted for *static* optical bypass in our IP over WDM reference design and showed a 10-15% decrease in Capex. We have also shown that this gain levels off as we add more bypass.

- Next, we outlined a design procedure that modeled a converged packet-circuit network based on our unified control architecture. Overall Capex and Opex savings of nearly 60% and 40% respectively are achieved in comparison to today's IP-over-WDM core networks. Furthermore such savings are found to be insensitive to varying traffic-matrices; and scale better (at a rate of $11m/Tbps vs. $26m/Tbps) as the overall traffic grows to five times the original aggregate.

Introduced Map-Abstraction into MPLS based Networks: We have mentioned before that MPLS networks have the flow-abstraction but lack the map-abstraction. We further validated our architectural approach, by introducing the map-abstraction into MPLS networks, and replicating services offered by MPLS today.

- We identified how we can replace *all* MPLS control plane functionality like signaling (RSVP) and routing (OSPF) within a controller's domain by writing network applications on top of OpenFlow/NOX. We have replicated discovery, recovery, label distribution, bandwidth reservation, and admission control via Constrained SPF calculations, while still using the standard MPLS data-plane.
- We built another prototype to demonstrate an MPLS - Traffic Engineering service that traffic engineered LSPs based on bandwidth-reservation and admission control. We have also shown how our TE-LSPs can have all the features they have today such as auto-bandwidth, priority, and explicit routes. Our solution again involved 2 orders of magnitude lesser line-of-code compared to the existing MPLS control.
- Finally we have identified opportunities where our control architecture can potentially solve problems that the existing MPLS control cannot.

1.6 Organization of Thesis

This chapter is essentially an extended summary of the thesis. We briefly described the significant differences in IP and transport network architectures, how they are separately designed and controlled today, and then defined the problem statement as one where we need to find a way to run one network instead of two. We discussed an alternative viewpoint in which the goal of running one network can be achieved by eliminating circuit switching in transport networks; but showed why both packets and circuits belong in future networks and a better idea would be to converge their operation.

We proposed our solution to convergence – unified control architecture – as a combination of two control abstractions: a common-flow abstraction and a common-map abstraction. We showed how the former fits well with both types of network and provides a common paradigm for control, while the latter makes it easy to insert new functionality into the network. We briefly discussed a previous attempt at unified control (GMPLS) and identified reasons for its failure, the fundamental one being the lack of a map abstraction. And finally we summarized our contributions in the previous section. The rest of this thesis is organized as follows.

In Chapter 2, we describe the common-flow and common-map abstractions in more detail. We describe how packets and circuits can be abstracted as flows; and then delve into abstractions for different kinds of circuit switches and requirements for a common-switch API. Next we detail the representation, construction and maintenance of a common-map as well as the requirements of a common-network-API. We present three prototypes (named pac.c) we built to validate our architectural and control plane constructs. We explore the extensions we have made to the OpenFlow interface to create a common API for both kinds of switches, and the changes we made to a network-operating-system (NOX) to have it present a common-map and network-API to network-control-functions.

In Chapter 3, we demonstrate the simplicity and extensibility of our proposed unified control architecture. First we demonstrate an implementation of a control function across packets-and-circuits using our full pac.c prototype in an emulated-WAN structure. Then we compare our implementation to one which would use existing control-solutions in the industry today. Then we give examples of more new control applications across packet and circuits, and show how our work is far more extensible than existing control-solution in the industry. Finally we discuss solutions to three deployment challenges faced by *any* unified control solution for packet and circuit networks.

In Chapter 4, we give a detailed example of today's IP over WDM design methodology, which we model as a reference design. We then propose a core network that benefits from both packet-switching and dynamic circuit switching under an SDN based unified control architecture. We perform a comprehensive Total Cost of Ownership (TCO) analysis to judge the economic impact of our proposed changes. More importantly, we provide technical solutions to practical issues that have hampered the adoption of some of these ideas in the recent past.

In Chapter 5, we show how existing MPLS applications and services can be offered by an IP/MPLS network based on our control architecture. We show that by introducing the map-abstraction and retaining the MPLS data plane (flow abstraction) we can replace all MPLS control-plane functionality. We present implementation details of our prototype system where we have shown applications like MPLS Traffic Engineering on top of the map abstraction. Finally, we discuss how introducing the map-abstraction in MPLS networks fits well with our unified-control architecture for packet and circuit networks- a fact that makes our control architecture ideal for multi-layer networks

We conclude in Chapter 6, and present related work and future research directions in this area.

Chapter 2

Architecture & pac.c Prototypes

In the previous chapter, we introduced the concept of a unified control architecture for the converged operation of packet and circuit switched networks [11]. Our control architecture (Fig. 2.1) is based on two abstractions: the common-flow abstraction and the common-map abstraction.

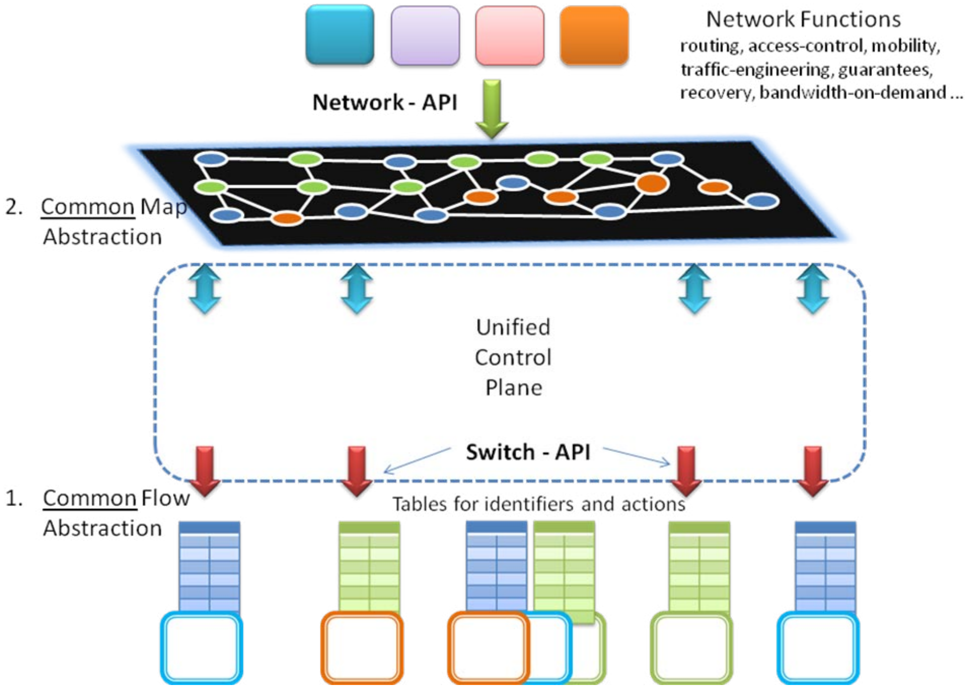

Network Functions
routing, access-control, mobility, traffic-engineering, guarantees, recovery, bandwidth-on-demand ...
Network - API
2. Common Map Abstraction
Unified Control Plane
Switch - API
1. Common Flow Abstraction
Tables for identifiers and actions

Figure 2.1: Unified Control Architecture

The common-flow abstraction is based on a data-abstraction of switch *flow tables* manipulated by a *switch-API*. The flow tables take the form of lookup-tables in packet switches and cross-connect tables in circuit switches; and together with the common switch-API, abstract away layer and vendor specific hardware and interfaces.

The common-map abstraction is based on a data-abstraction of a network-wide *common-map* manipulated by a *network-API*. The common-map has full visibility into both packet and circuit switched networks, and allows creation of network-applications that work across packets and circuits. Implementing network-functions in such applications is simple and extensible, because the common-map abstraction hides the details of state distribution from the applications.

In this chapter we give details of the design of the two abstractions. We discuss the flow abstraction as applied to packet-switches [12] and then give design-details on what is needed to apply such an abstraction to circuit-switches [13]. Further we show how a common-switch API can be designed to manipulate this data-abstraction. For the common-map abstraction, we give design details on how we represent such a map, how it can be built and maintained, and briefly what a network API that manipulates such a map could look like.

Next we present implementations of our architectural approach. We developed prototypes to help us validate our architectural constructs and improve on their implementation. We give details of three prototypes systems we built (and named **pac.c** for **p**acket **a**nd **c**ircuit **.c**onvergence). Two early prototypes helped us design and validate the circuit-switch abstraction and common switch-API. Accordingly they use different kinds of circuit switches – a wavelength switch and a time-slot based switch – together with packet switches. The third, more complete prototype helped us understand the intricacies of building a converged packet-circuit network with a common-map and network-API. We discuss the ideas we demonstrated and the lessons we learned with each prototype.

2.1 The Common-Flow Abstraction

The objective is to develop a generic data-plane abstraction based on *flows*, for all kinds of packet and circuit switches. We first discuss packet-switches and then extend those ideas to circuit switches to develop a common-flow abstraction and switch-API.

2.1.1 Packet-switching and the Flow Abstraction

Our discussion of the packet-flows is based on a generic packet-switch abstraction that is implicit in the design of the OpenFlow protocol [12, 14]. Here we wish to give an overview of the abstraction and the reasoning behind it.

We start by discussing the various kinds of packet-switches used today and their common characteristics. We then discuss a generalization of these switches into a single representation; and detail the relationship of that representation with the concept of a 'flow', both within a switch as well as across multiple switches in the network. Finally we outline the main functions of an interface (a switch-API) used to manipulate such a representation.

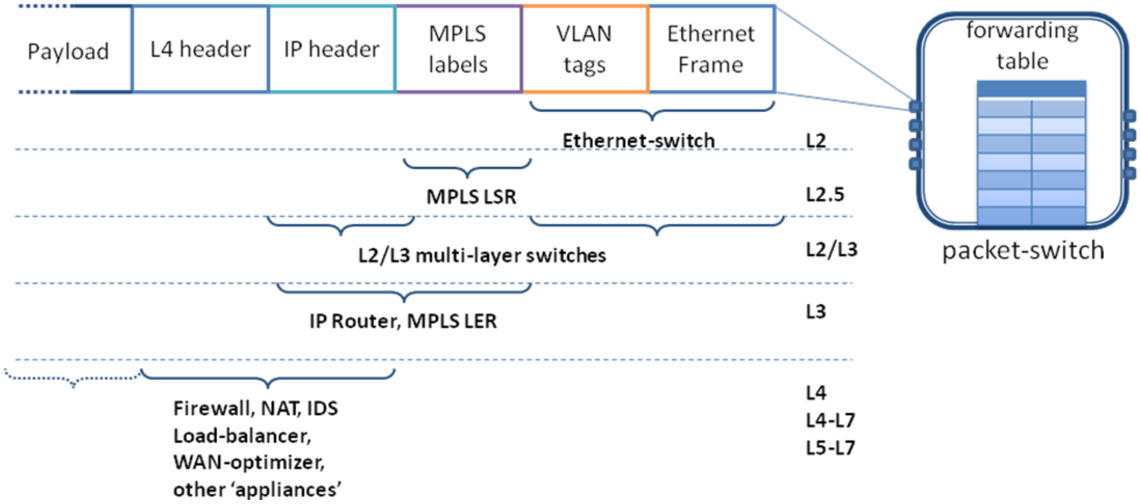


Figure 2.2: Different kinds of packet-switches

Packet-Switching: Switching in packet-networks is often defined in a layer-specific way. The network layer (or Layer 3 or L3) originally represented the switching (or routing[*]) layer. But switching exists in the lower layers (L2), higher layers (L4-L7), and new intermediate layers (L2.5) have been coined. And so it is worth noting that today a) the layer terminology simply refers to the different parts of the packet-header; and b) packet switches in different layers make forwarding decisions based on the layer they are part of (Fig. 2.2). For example:

- An IP router (L3) may typically forwards a packet based on the IP destination address in the IP header of the packet.
- A traditional Ethernet bridge (L2) forwards packets based on MAC addresses and VLAN ids. If the bridge includes 'routing' functionality i.e if it looks at IP packets to route between VLANs, it becomes a L2/L3 multi-layer switch.
- Some routers forward packets based on 'tags' such as MPLS labels. Since the label is inserted in a packet typically between a MAC header (L2) and an IP header (L3), it is referred to as L2.5 switching.
- And there are other, more special-purpose switches (or appliances or middleware) that forward packets (forward, drop, modify-and-forward etc.) based on yet other parts of the packet header.

And so, it is clear that irrespective of the type of packet switch, all of them perform the same basic functionality of identifying the part of the packet-header they are interested in; matching that part to related-identifiers in a lookup-table (or forwarding table); and obtaining the decision on what-to-do with that packet from the entry it matches in the lookup-table. Some other aspects of packet-switching in today's networks:

- In most cases packets are switched independently *within a switch,* without regard to what packets came earlier or which ones might come afterwards. Each packet is dealt with in isolation while ignoring the logical-association between packets that are part of the same communication[^].

[*] Routing is just another name for switching in L3, just like bridging is another name for switching in L2.
[^] Examples of such communications and logical-associations were discussed in Chapter 1 (Table 1.1).

- As packets travel from switch to switch, each switch makes its own independent forwarding-decision. Packets correctly reach their destination because the switches base their forwarding decision on some form of coordination mechanism between the switches. But such coordination mechanisms typically tend to be a) restricted to the layer/network in which the switch operates; and b) only give information for the part of the packet-header related to that network. For example,
  - STP co-ordinates Ethernet networks by preventing loops in the network topology, so that switches can learn about destinations *only* from packets coming in from un-blocked ports, and also forward packets to only those un-blocked ports.
  - IP networks use completely different coordination-mechanisms (routing protocols OSPF, IS-IS, I-BGP) that only refer to IP-destination prefixes.
  - MPLS networks use LDP or RSVP as co-ordination mechanisms similar to signaling. Here too the co-ordination is layer specific – bindings of MPLS labels to IP addresses.
- Finally, because packets are switched in isolation within and across switches, and the logical association between packets is typically not processed; it becomes very hard to perform accounting and resource management in a packet network. For example, if it is difficult to get a common handle for a stream of packets between two servers travelling across an Ethernet network; it is very difficult to tell which how much bandwidth the stream is consuming (accounting); or make resource decisions for just that stream (a specific path, bandwidth-reservation, etc.).

Packet-Flow Abstraction: From the previous discussion, it is clear that there are advantages to defining a *generic* (layer independent) data-plane abstraction for packet-switches based on 'flows':

- A packet-flow is a logical association (or classification) between packets that are part of the same communication and are given the same treatment in the network (within a switch and from switch-to-switch);

- The data-abstraction is the representation of a packet-switch as flow-tables, ports and queues. The flow (or the logical-association) is defined in the flow-tables which have the ability to indentify the flow generically (in a layer independent way) from multiple parts of the packet header. For example:
  - If the logical association is simply a destination MAC or IP address, then the generic flow table should be able to behave like layer-specific switch-tables (eg. L2-tables in Ethernet switches or L3 tables in IP routers);
  - But if the logical association requires a mix of packet-header fields for identification, the table should be able to process this as well (for example the flow identification could require a mix of IP and TCP fields).
- Once the logical association has been identified, then all packets that have the same association are treated the same way *within* the switch; where the flow-table applies the same set of actions on all packets that are identified as part of the same flow.
- Furthermore, each switch that 'sees' the flow, does not make an independent, isolated decision on the treatment to the same flow. The decision on the treatment given to all packets in the flow is communicated in a layer-independent way to all switches through which packets in the flow traverse;
- The flow-definition serves as a common-handle with which accounting and resource-management in the network can be performed on a flow-level (not packet-level).
- Finally, the data-abstractions in all switches are manipulated by a layer-independent switch-API, which we discuss next.

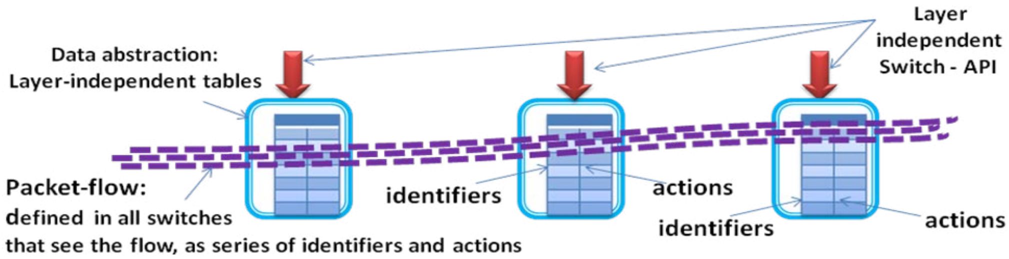

Layer independent Switch - API
Data abstraction: Layer-independent tables
Packet-flow: defined in all switches that see the flow, as series of identifiers and actions
identifiers
actions
identifiers
actions

Figure 2.3: Packet-flows

Switch API: With the aforementioned definition of packet-flows in switches based on data abstractions of <flow-tables, ports, queues>; any entity that makes decisions on flows needs a layer-independent switch-API to manipulate the data-abstractions. For example, such an entity decides on what constitutes a flow (the logical association); determines how to identify the flow (packet-header fields); and how to treat all packets that match the flow-definition in all switches that the packets flow through. In order to enable the entity to make these decisions, it needs to a) understand the capabilities of the data-abstractions (the flow-tables, ports, queues) and have control over its configurable parameters; b) have full control over the forwarding path; and c) have the ability to monitor or be notified of changes in switch state. Thus the layer independent switch-API includes the following set of functions:

- Get/Set Capabilities and Configuration:
  - Methods to get the representation of the switch as data-abstractions: ports, queues, tables and their features.
  - Methods to get the capabilities of the flow-tables: for example, the flows-identifiers and actions the table can process.
  - Methods to set configurable parameters for the data-abstractions: a) locally on port, tables or queues; and b) globally on a switch level.
  - Methods to query default or current values of these configurable parameters.
- Control forwarding state:
  - Method for adding flow definitions by specifying the flow-identifiers and related set of actions. Method for deleting the flow definition.
  - Method for changing the actions applied to a flow-definition.
  - Methods to set advanced forwarding state eg. logical ports.
- Monitor: Statistics and Status
  - Methods for querying flow-table state and flow statistics.
  - Methods for querying tables, port, queue and switch statistics.
  - Methods for setting traps for change in switch-state.

The methods described above may or may not elicit a response from the switch. In some cases, there are explicit-replies; for example - to request methods (get). In other cases there may be explicit or silent acknowledgements, or replies to indicate errors.

2.1.2 Circuit-switching and the Flow Abstraction

Circuit-Switching: Like packet-switches, circuit-switches can also be defined in layer-specific ways (Fig. 2.4). For example, time-slot switching, using SONET/SDH or OTN standards, are often described as Layer 1 (or L1) or physical-layer switching. Additionally, wavelength or fiber switches are described as L0 switches, not because a new layer has been included in the OSI model, but because it is a convenient way to describe switching at an even coarser physical granularity than time-slots.

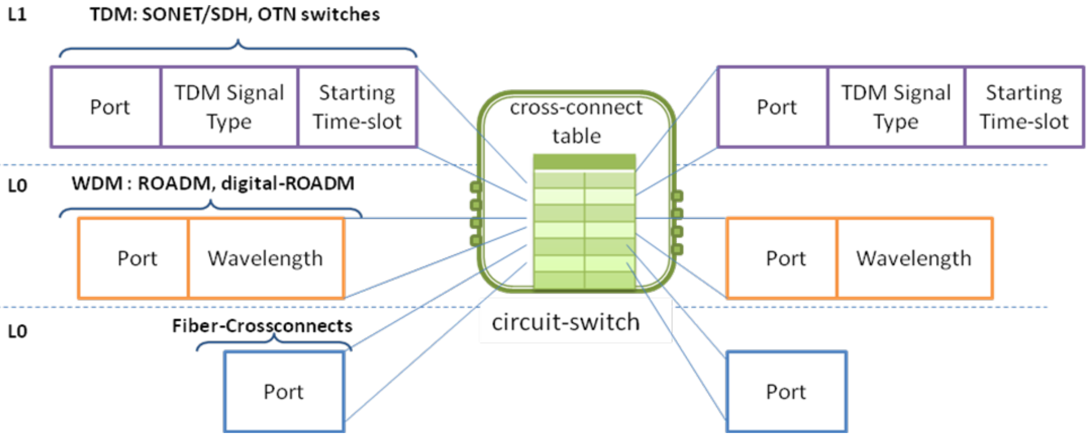


Figure 2.4: Different kinds of circuit switches

Nevertheless, all circuit switches maintain forwarding tables in the form of cross-connect tables with entries that are suited to the switching-type of the switch. In the following discussion, we consider two kinds of circuit switches - TDM and WDM; describe the creation of a circuit in each layer; and then relate a circuit to a 'flow'. We then develop the circuit-flow abstraction; show how to map packet-flows to circuit-flows and describe a common switch-API can be used to manipulate both data-abstractions.

TDM switching: A time-slot based switch has a time-synchronous switching fabric. It cross-connects time-slots on incoming ports to time-slots on outgoing ports. The following factors need to be considered for TDM switches[†]:

- Framing: There are different framing standards for TDM switch ports – SONET, SDH and OTN – frames differ in sizes and overhead bytes
- Line-rate: In the SONET standard, an interface with ~ 10Gbps line-rate (actually 9.95328 Gbps) is an OC192, but in OTN it is OTU2 (10.709225 Gbps).
- Time-slots: In an OC192, the number '192' comes from the fact that the line-rate can be divided into 192 time-slots each with a data-rate of roughly 50 Mbps.
- Signal-type: The smallest signal is the 50Mbps time-slot (STS-1). But larger signals are defined which use more time-slots. Thus the OC192 can carry multiple signals concurrently – 192 STS-1s; or 20 STS-1s and 10 STS-3s with 142 unused time-slots; or even 1 big STS-192c. Thus a SONET 'signal' is defined by the number of time-slots it uses and its starting (lowest) time-slot in an optical carrier (OCx).
- Concatenation: In SONET/SDH, signals can further be concatenated using contiguous (STS-3c, STS-12c, VC-4-4c etc.) or virtual concatenation (VCAT, ODUflex);
- Switching-granularity: It is necessary to understand the minimum switching granularity of the switching fabric – for example, if it has the ability to switch time-slots as small as STS-1s.

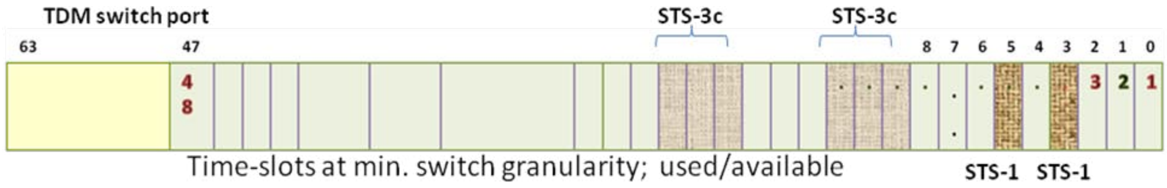


Figure 2.5: Representing bandwidth in TDM switches

- Bandwidth representation: Given the line-rate & the minimum switching granularity, a bandwidth representation of a time-slotted port can be drawn up (Fig. 2.6). It is not enough to give cumulative numbers for reserved and un-reserved bandwidth; instead

[†] Other factors such as signal-multipliers, transparency, and rules for contiguous and virtual concatenation have been left out for clarity

each minimum-granularity time-slot on a port can be represented by a bit field in a bit-map. Then a 1 or 0 value for the bit signifies the availability of that time-slot.

- Cross-connection: To specify a cross-connection, the input and output time-slots have to be specified (Fig. 2.4). The time-slots can be specified as a 3-tuple of <port, TDM signal-type, starting-time-slot>. The port identifies the physical port; the signal-type identifies the number of time-slots; and the starting-time-slot identifies the lowest (or earliest-in-time) time-slot in the carrier where the signal starts. In Fig. 2.5, one of the STS-3cs starts at time-slot #10 (bit number 9) in the OC-48 the bit map represents, and since it is an STS-3c signal it occupies 3 time-slots (10-12). In the same example, we see that the carrier currently has 2 STS-1s and 2 STS-3cs (shaded) and 40 free (unused) time-slots.
- TDM circuit: Therefore a circuit in a TDM network is simply a series of cross-connections in switches. As an example, consider the STS-3c circuit (150 Mbps) in Fig. 2.6. The signal-type (STS-3c) must remain the same throughout the circuit definition. The port numbers have significance only to the switch the port belongs. And the time-slots can interchange in a connection if the switch supports such behavior. But the time-slots on a link must be the same. For example, the outgoing STS-3c signal on the first switch starts on the 10th time-slot (on port 4). Therefore the cross-connection on the second switch must specify the same start-time-slot on port 5, which connects to port 4 on the first switch.
- Bi-directionality: Circuits are always bidirectional. Thus specifying a cross-connection from an 'in' 3-tuple to an 'out' 3-tuple, simultaneously specifies exactly the same cross-connection in the reverse direction.

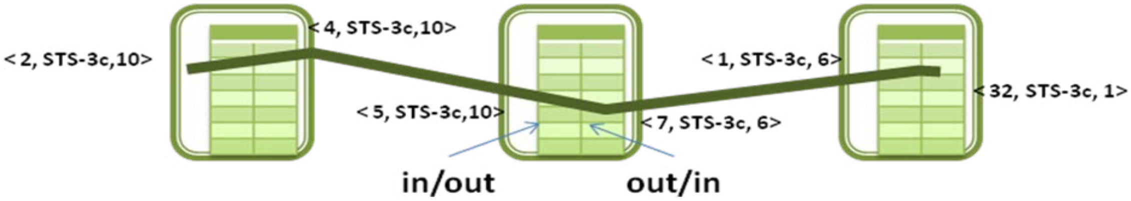


Figure 2.6: TDM circuit

WDM switching: A wavelength switch cross-connects an incoming wavelength (or set of wavelengths) on a port connected to an outgoing same-or different wavelength (or set of waves) on a different port. Typically it is implemented as a combination of a wavelength-demultiplexer, a switching-fabric that switches light-beams and a wavelength multiplexer. The following factors need to be considered for WDM switches:

- Switching-granularity: Does the switch support a minimum-switching granularity of a single wavelength or band of wavelengths. If it is the latter, then how is the band defined (in terms of the number of wavelengths). A fiber-switch may be considered as a special case of a wavelength switch, where the ‘band’ is defined as all the wavelengths that can be supported in the fiber.
- Line-rate: A wavelength switch has mux/demux filters designed to operate in at a certain line-rate, which must be specified. Optical fiber-switches are typically agnostic to whatever signals are carried in the incoming fiber. However, there may be transponders or WDM filters attached to the ports that may limit the switch to a single line-rate. Then the line-rate and wavelengths of the signals have to be considered.
- Fabric-technology: For WDM switches, depending on whether a wavelength is switched electronically or optically, a number of additional factors crop up. If the switching fabric is electronic, the framing used has to be taken into consideration. Also the incoming wavelength can be switched to a different out-going wavelength. There may also be other technology dependant feature support – eg. out-going wavelength tunability; variable line rate; optical supervisory channel (OSC) support.
- Bandwidth representation: A wavelength switch port can be represented as a bit-map. The bits in the bit-map represent the ITU grid frequencies [30]. Flags can be used to identify the spacing of the frequency channels – 25 GHz, 50 GHz, 100GHz etc.; and to identify a C, L or S band system. Fig. 2.7 shows a 100 GHz spaced C-band system (191.3 THz to 196.7 THz). Using multiple bit-maps, the switch can report the waves it supports on a port, and the ones that are currently cross-connected (in-use).

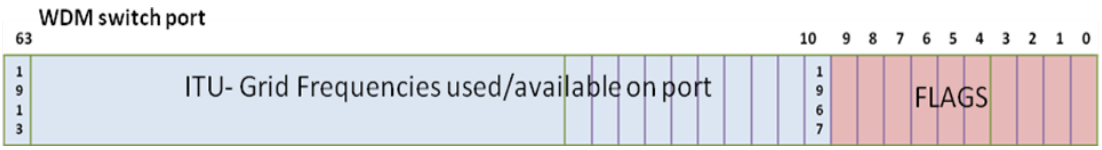


Figure 2.7: Representing bandwidth in WDM switches

- Cross-connections: To specify a cross-connection, the input and output wavelengths have to be specified (Fig. 2.4). The wavelengths can be specified as a 2-tuple of <port, ITU-grid-frequency-numbers>. The port identifies the physical port; the wavelength or a set of contiguous wavelengths (for a waveband) on the physical port are identified by their corresponding bits in the relevant ITU-grid-bitmap. All other technology dependant factors have to be (implicitly) considered when specifying this cross-connection. For example, two different wavelengths cannot be cross-connected when wavelength conversion is not supported; if there are transponders or wavelength filters then different line-rates should not be cross-connected; wavebands have to be of the same size (same number of lambdas) etc.
- Wavelength circuit: A circuit in a WDM network is then simply a series of cross-connections in switches. Fig. 2.8 shows a single-wavelength circuit (not waveband) in a network of electronically-switched wavelength switches, where each hop of the circuit comprises of a different wavelength.

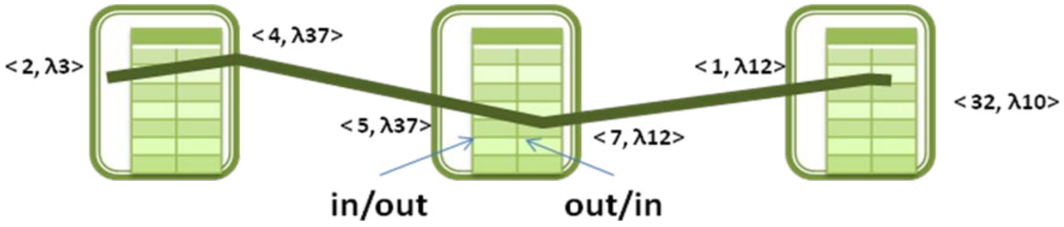


Figure 2.8: Wavelength circuit

- Bi-directionality: Similar to TDM circuits, wavelength-circuits or fiber-circuits are always bidirectional. Specifying a cross-connection from an in-lambda to an out-lambda simultaneously specifies exactly the same cross-connection in the reverse direction.

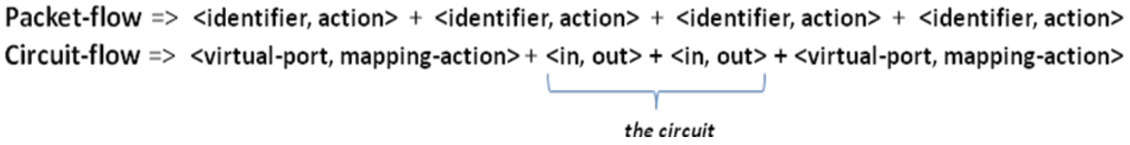
Viewing a Circuit as a Flow: It is worth comparing Figs. 2.6 and 2.8 which show TDM and wavelength circuits, to Fig. 2.3 which shows a packet flow. We note the following similarities:

- Both packet switches and circuit switches can be represented as forwarding-tables that support translations – in the packet case, an incoming <packet, port> is translated to an outgoing <packet′, port′>; in the circuit case, an incoming < λ, time-slot, port> is translated to an outgoing <λ′, time-slot′, port′>.
- A packet-flow is series of <match-identifiers, actions> in all packet-switches that the flow traverses; Likewise a circuit is a series of <in-x-tuple, out-x-tuple> cross-connections in all the switches the circuit passes through. Furthermore a decision to create a cross-connection within a switch is not-independent of cross-connections in other switches that make up the circuit (similar to our definition of packet flows).

And so, given these similarities, it is easy to see that the data-abstraction and switch-API that we discussed in the previous section can potentially be used in similar ways in the circuit-switching context.

There is however one important distinction. A packet-flow is the logical-association between packets that are part of the same communication. The packet-flow exists *in itself* to bind-together the packets as they flow in the network. On the other hand, a circuit as defined so far is a *carrier*; and it only becomes a logical-association for something between two-communication end-points, when that something is mapped into the circuit.

And so, to complete our analogy to packet-flows, a circuit becomes a *circuit-flow* only when we account for the end-points of the circuit in relation to what gets mapped into the circuit. Such an end-point is represented as a virtual-port with associated mapping-actions.



Packet-flow => <identifier, action> + <identifier, action> + <identifier, action> + <identifier, action>

Circuit-flow => <virtual-port, mapping-action> + <in, out> + <in, out> + <virtual-port, mapping-action>

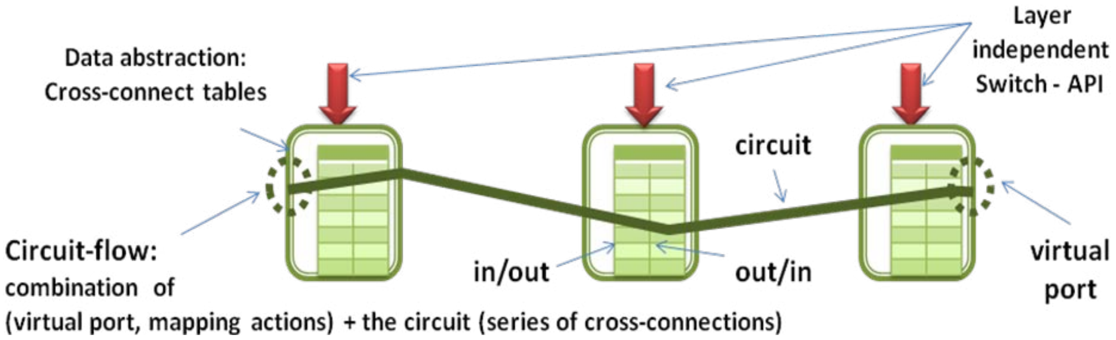

Layer independent Switch - API
Data abstraction: Cross-connect tables
circuit
Circuit-flow: combination of (virtual port, mapping actions) + the circuit (series of cross-connections)
in/out
out/in
virtual port

Figure 2.9: Circuit-flows

Circuit-Flow Abstraction: We are now ready to define a generic data-plane abstraction of circuit switches based on flows:

- A circuit-flow is a logical association between a payload that is carried by a circuit and is therefore given the same treatment in the network (from one end of the circuit to the other);
- The data-abstraction is the representation of a circuit switch as flow-tables and ports: the flow is defined by a) virtual-ports that identify the end-points of the circuit with mapping-actions to map payloads into-and-out of the circuit; and b) a bidirectional cross-connection which translates an incoming circuit-identifier to an out-going circuit-identifier;
- The decision on the treatment given to the payload in the circuit-flow is communicated in a layer-independent way to all switches which 'see' the flow, ie. the switches through which the circuit passes;
- The circuit-flow definition serves as a common-handle with which accounting and resource-management in the network is performed on the circuit-level.
- Finally, the data-abstraction in all circuit-switches is manipulated by a layer-independent switch-API. Given the similarities between the data-abstraction for packet and circuit switches, it is easy to see that the API can be a common one for both kinds of data-abstractions.

Mapping Packet-flows to Circuit-flows: Representing circuit-flows as combinations of virtual-ports and cross-connections presents a way to map packet-flows to circuit-flows, irrespective of the way in which packet and circuit switches are interconnected. Consider the different ways in which packet and circuit switches can be interconnected. In Fig. 2.10,

- If the packet switch P is connected to the TDM circuit switch via a Packet over SONET (POS) interface, then the virtual port is manifested by the PoS port.
- If P is connected to a hybrid switch (with both packet/circuit switching fabrics) via an Ethernet interface (ETH), then the hybrid switch has the capability to adapt Ethernet frames to TDM frames (SONET or OTN). The virtual port is then manifested by a mapper that performs this mapping.

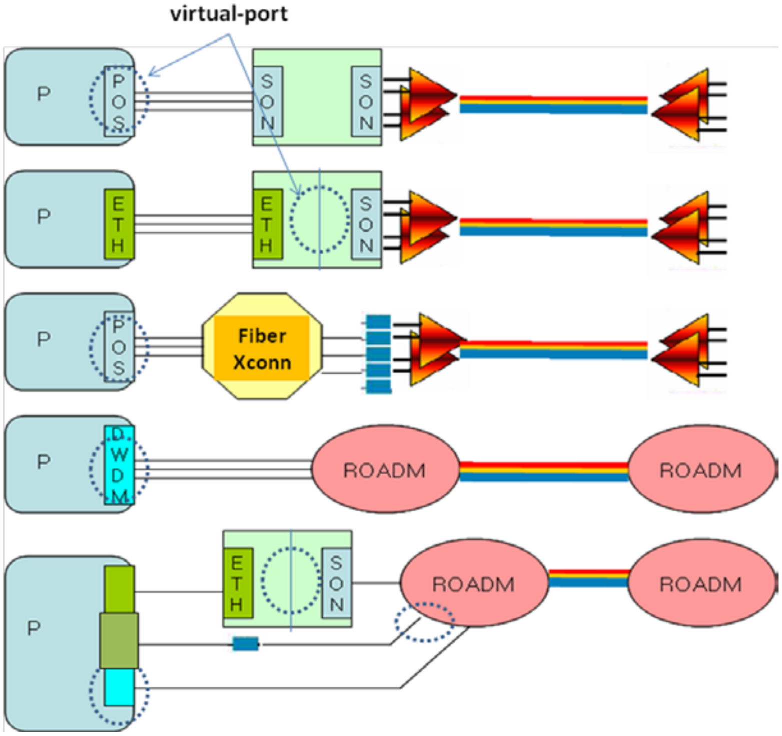


Figure 2.10: Various ways to interconnect packet and circuit switches

- If P is connected via a fiber cross-connect (Xconn) to a DWDM line-system, then again the POS port is an instance of the virtual-port. This PoS port typically does not use high quality transceivers needed for long distance communications neither does it use the standardized ITU grid wavelengths– hence DWDM transponders are needed before the signal is transmitted over the DWDM line systems. Such transponders are reported as technology-specific switch-features by the Fiber-Xconn.
- If P uses DWDM transceivers (with suitable framing), it could connect to the DWDM line-system via a wavelength-switch (ROADM). The virtual port is the DWDM interface on the packet switch. The wavelength on the transmitters may even be tunable. If instead P uses an Eth interface with a transponder to connect to the ROADM, the virtual port would be the transponder in the ROADM interface.
- Lastly, different ports on the packet switch could use different combinations of the above, in which case the virtual ports for each of the circuit-flows could be in different switches.

To map a packet-flow into a circuit- flow (Fig. 2.11), the last action in the action-set defined for a packet-flow, forwards all packets that match the packet-flow definition to the virtual-port. This way multiple packet-flows can be mapped into the same circuit-flow. At the other-end of the circuit-flow, the packet-flow identifiers match on the virtual-port and any other packet-header-fields to distinguish between and de-multiplex the packet-flows coming out of the virtual port.

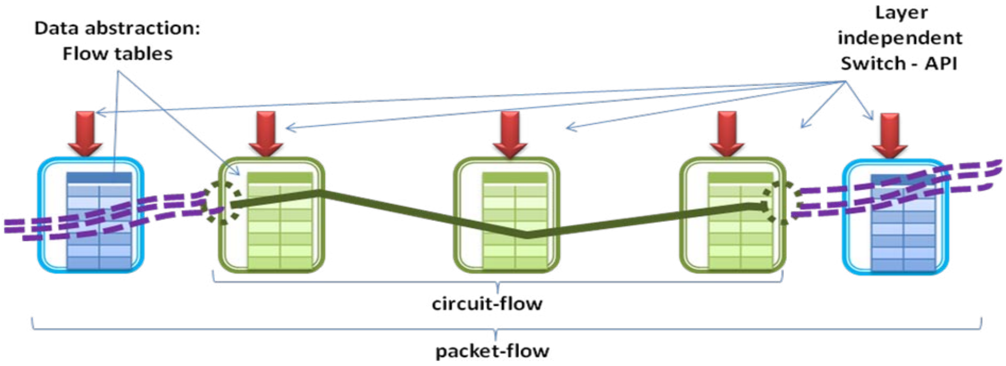


Figure 2.11: Mapping packet-flows to circuit-flows (and back)

Switch API: The circuit-flow abstraction holds remarkable similarities to the packet-flow abstraction. Unsurprisingly, the switch-API that manipulates a circuit switch abstracted as <flow-tables, ports>, also has a similar set of functions – the ability to understand the capabilities of the data-abstraction and have control over its configuration; full control over the forwarding state; and the ability to monitor status and obtain statistics. In fact a common switch-API can be developed for both packet and circuit switches, with small modifications to account for technology-dependant features. We details such modifications below:

- Get/Set Capabilities and Configuration:
    - Methods to get the capabilities of the cross-connect-tables. Features include:
        - Switching fabric type – time-slot (TDM), wavelength (WDM) or fiber;
        - Switching-granularity – TDM: smallest signal-type that can be switched; WDM: single wavelengths or band of wavelengths;
        - TDM signal support, WDM wavelength range
        - Mapping-Actions support: TDM: virtual-concatenation support using VCAT (SONET/SDH) or ODUflex (OTN), LCAS support etc; WDM: variable line-rate support, tunable wavelength support etc.
    - Methods to get the switch make-up: port, tables and bandwidth representations.
        - Port representations – line rates (OC48, OC192, OTU0/1/2/3 etc.), framing types (SONET, OTN, Ethernet etc.)
        - Bandwidth representations – TDM ports: used and available time-slots at minimum switch-granularity; WDM ports: used and available wavelengths for a defined grid spacing and range
        - Recovery support – most circuit switches have in-built hardware support for link recovery. Such recovery types are known as 1+1, 1:1, N:1 etc.
        - Neighbor discovery support – some circuit-switches (digital ones) have the ability to discover their neighbors on the links they share with them.

        - ▪ Technology dependant switch features –eg. WDM: wavelength conversion capabilities; transponders; WDM filters etc.
    - Methods to set configurable parameters on a local port or table basis as well as on a global switch basis.
    - Method to query default or current values of these configurable parameters.
- Control forwarding state:
    - Method for adding circuit-flow definitions by specifying the virtual-port and mapping-actions.
    - Method for creating cross-connections between: physical fiber-ports, wavelengths and time-slots. Also means for connecting virtual-ports to any physical fiber-port, wavelength or time-slot.
    - Method for changing the mapping-actions applied to a circuit-flow's virtual-port.
    - Method for deleting the circuit-flow definition – both cross-connections and virtual-ports. Deleting a virtual port is equivalent to deleting the cross-connections that support it within a switch.
    - Methods to set recovery state eg. link-protection.
- Monitor: Statistics and Status
    - Methods for querying cross-connect table state
    - Methods for querying virtual-port state (example: the packet-flows mapped into the port) and statistics (transmitted/received bytes etc.).
    - Methods for setting traps for change in switch-state: example physical port up/down, or virtual-port queue depth (queue of packets entering circuit)

As before, the methods described above may elicit a reply from the switch with requested information, positive acknowledgements or error messages, or the switch may simply process the function with silent acknowledgements.

In our work, we have implemented the common-switch-API by extending the OpenFlow protocol to manipulate circuit switch flow-tables [36].

2.2 The Common-Map Abstraction

The common-map abstraction is based on a data-abstraction of a network-wide *common-map* manipulated by a *network-API* (Fig. 2.12a). The common-map has full visibility into both packet and circuit switched networks, and allows the creation of network-applications that work across packets and circuits. The common-map is created and kept updated and consistent with network state by the unified-control-plane (UCP).

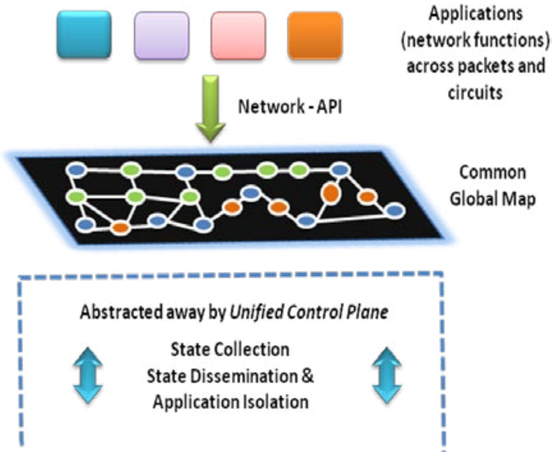

Applications (network functions) across packets and circuits
Network - API
Common Global Map
Abstracted away by *Unified Control Plane*
State Collection
State Dissemination &
Application Isolation

Figure 2.12(a): Common-Map Abstraction (same as Fig. 1.12)

The common-map makes it simpler to implement network functions by abstracting away (hiding) the means by which network state is collected and disseminated. Today network functions are implemented as distributed applications tightly coupled to the state-distribution mechanisms that support them. By breaking this coupling, the common-map abstraction allows applications to be implemented in a centralized manner. Not only does this make the applications simpler, it also improves extensibility, as inserting new functions into the network becomes simpler. Moreover, the network itself becomes programmable, where functionality does not have to be defined up-front by baking it into the infrastructure. A network-API can be used to write programs that introduce new control-functionality as the need for it arises. With its full visibility, the common-map allows new features to be supported that take advantage of both packets and circuits.

Importantly it gives the application-programmer the choice to treat packets and circuits together or as separate layers, or even completely ignore one or the other in the application context. Together the common-map abstraction benefits of programmability, simplicity, extensibility and choice ease the path to convergence and innovation in packet and circuit networks.

In the following discussion we first briefly describe a representation of the common-map. We then describe means by which it can be constructed and maintained, with emphasis on the important aspects of link-discovery and layering. Finally we detail possible features of the network-API.

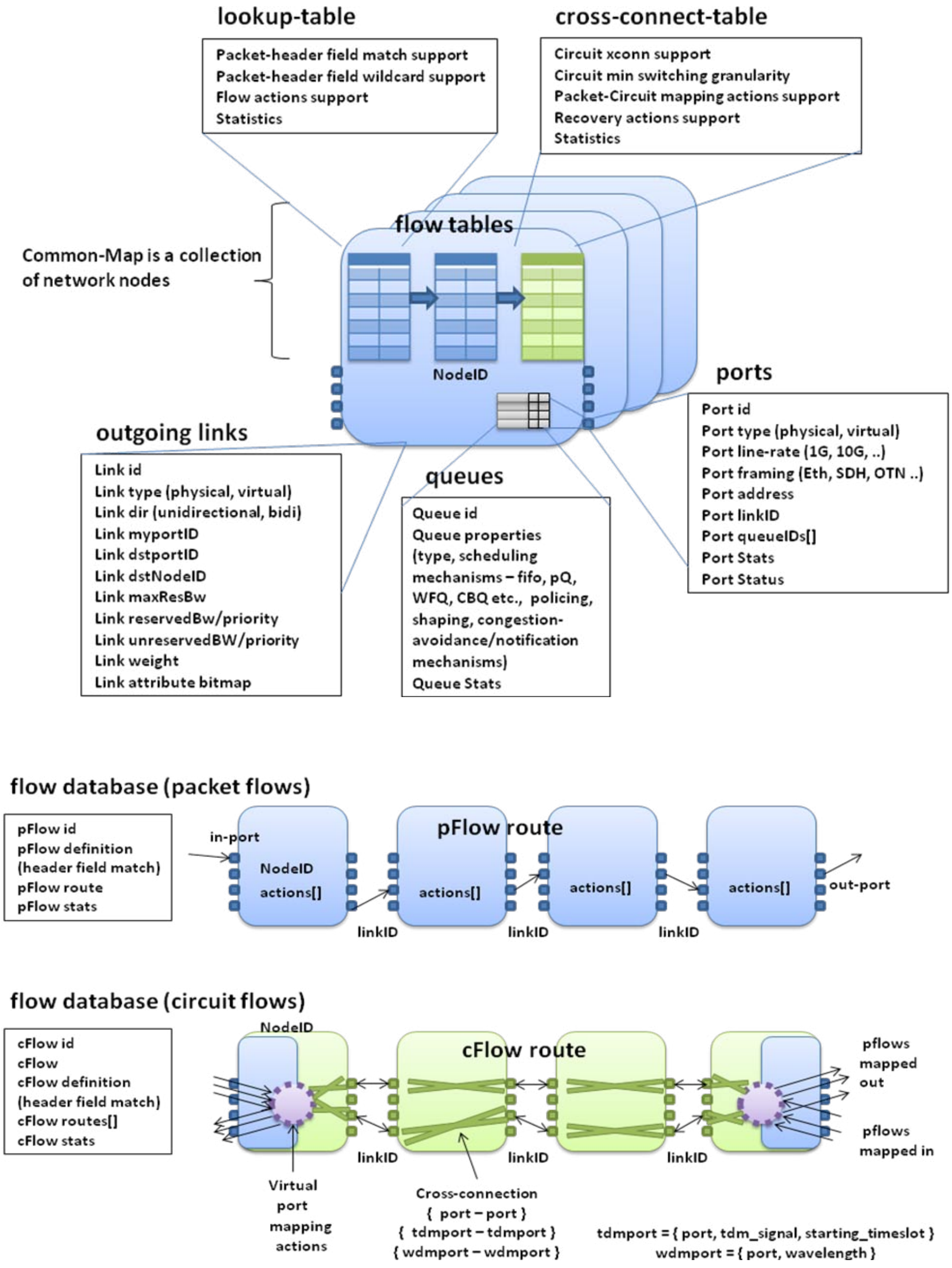
2.2.1 Common-Map Representation

In Chapter 1, we introduced the common-map as an annotated graph (or database) of the network topology (Fig. 2.12b). This graph is a collection of network Nodes. In the context of wide area networks, the nodes are essentially switches[†]; packet, circuit, and hybrid-switches which have both packet and circuit switching features.

Nodes: Each node is a collection of the following: flow-tables (both lookup-tables and cross-connect tables); outgoing links (physical and virtual); ports (physical and virtual); and queues. The node is a data-structure that represents the switch abstracted as <tables, ports, queues>. In itself there is not much information regarding the node other than a unique identifier such as a Node-id. However, the collections held within a Node give more details of the data-abstraction.

Ports: The port data-structure includes information on the type of port – either physical or virtual. It has a unique identifier (PortId) and one or more network addresses and names. A physical port can have meaningful line-rates and framing type and a LinkId for the physical Link for the connected link. It can also have a number of Queues attached it (or at least indexes into the Queue collection). A virtual-port can be of many types. It can represent the end-point of a circuit flow into which packet flows are mapped.

[†] Note that in other networking contexts such as enterprise or campus LANs, 'Nodes' can also include end-hosts or middleware connected to the switches.


lookup-table
Packet-header field match support
Packet-header field wildcard support
Flow actions support
Statistics
cross-connect-table
Circuit xconn support
Circuit min switching granularity
Packet-Circuit mapping actions support
Recovery actions support
Statistics
Common-Map is a collection of network nodes
flow tables
NodeID
ports
Port id
Port type (physical, virtual)
Port line-rate (1G, 10G, ..)
Port framing (Eth, SDH, OTN ..)
Port address
Port linkID
Port queueIDs[]
Port Stats
Port Status
outgoing links
Link id
Link type (physical, virtual)
Link dir (unidirectional, bidi)
Link myportID
Link dstportID
Link dstNodeID
Link maxResBw
Link reservedBw/priority
Link unreservedBW/priority
Link weight
Link attribute bitmap
queues
Queue id
Queue properties (type, scheduling mechanisms – fifo, pQ, WFQ, CBQ etc., policing, shaping, congestion-avoidance/notification mechanisms)
Queue Stats
flow database (packet flows)
pFlow id
pFlow definition (header field match)
pFlow route
pFlow stats
in-port
NodeID
actions[]
pFlow route
actions[]
actions[]
actions[]
out-port
linkID
linkID
linkID
flow database (circuit flows)
cFlow id
cFlow
cFlow definition (header field match)
cFlow routes[]
cFlow stats
NodeID
cFlow route
pflows mapped out
pflows mapped in
linkID
linkID
linkID
Virtual port mapping actions
Cross-connection
{ port – port }
{ tdmport – tdmport }
{ wdmport – wdmport }
tdmport = { port, tdm_signal, starting_timeslot }
wdmport = { port, wavelength }

2.12(b): Common-Map Databases (same as Fig. 1.10 and Fig. 1.11) I

In some cases it can be manifested by a physical port. In other cases, it can represent a group of physical-ports for a variety of purposes, such as broadcast or multipath load-balancing. Finally, port level statistics and status indicators are also maintained as part of this collection.

Tables: The flow-table information for the lookup (packet) and cross-connect (circuit) tables mainly includes information of the capabilities and features supported by the tables. For example, the lookup table matching and wildcarding support for packet-header-fields, and the actions that can be applied to a packet are stored here. For the cross-connect table, the particular kind of switching fabric, minimum switching granularity, mapping-actions and recovery types supported are stored here. Lastly statistics are maintained on a table basis.

Queues: The queue-data structure can include information such as the QueueId; the PortId of the port it is assigned to; Queue statistics; and any other information regarding the type of queue – min-rate, max-rate, associated scheduling mechanisms like FIFO, WFQ, priority queueing, class-based-queueing, policing and shaping support, congestion avoidance (RED) and other AQM support. It can also include information on traps set by the user to be notified for queue occupancies cross set thresholds.

Links: While links in the network can be maintained in a database separate from the nodes-database, there are many advantages to maintaining it as part of the nodes data-structure. For example, it is a convenient way to quickly get the outgoing links from a node which is required by many routing algorithms; it is also convenient to represent unidirectional links (for eg. unidirectional MPLS tunnels or other virtual-links) or link features which are different in either directions (such as bandwidth reservations in MPLS tunnels). The link data-structure includes information on the destination portId and nodeId. It also includes information of max bandwidth, reserved bandwidth, unreserved bandwidth, and actual bandwidth usage for the out-going direction. The actual bandwidth representation depends on the type of link – eg a packet-link (used for MPLS-TE) or a TDM circuit link or a WDM fiber link. In all cases the notion of bandwidth and its reservation holds. Additionally the links database can maintain a link-cost (or weight) useful in certain path-calculations, as well as a set of attributes that the user can define for similar purposes. Ultimately the links data-structure is a repository of all link information. It is up to the user to use-or-ignore parts of the structure in the context of the application or its corresponding networking domain.

2.2.2 Common-Map Construction & Maintenance

To create a common-map, the unified control-plane learns about switch ports, queues, tables, and other switch characteristics and capabilities using the switch-API (discussed in the previous section as Get/Set Capabilities and Configuration). However, the network database is incomplete without information on network *links*. Thus as an important part of the construction of the common-map, we discuss various kinds of packet and circuit link-discovery mechanisms.

Link Discovery: With reference to Fig. 2.13, we define a packet link to be one that interconnects interfaces on two packet switches (or the packet switching part of multi-layer switches). These interconnections could be physical (for example two Gigabit Ethernet ports inter-connected by an Ethernet cable) or they could be virtual (eg. two PoS or GE interfaces connected over the wide-area by an underlying circuit). Circuit links are *always* physical links – essentially fiber or wavelengths that interconnect interfaces on circuit switches (or the circuit switching part of multi-layer switches).

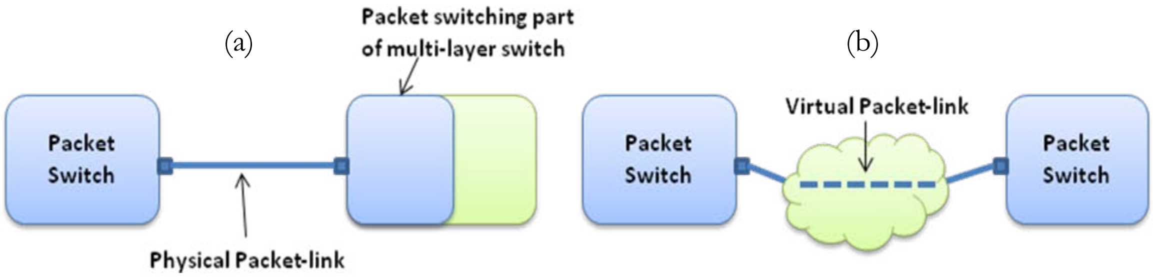


Figure 2.13: Physical and virtual packet links

Physical-packet links: Discovery can be performed by the control plane using test-packets. With the use of a mechanism to send packets selectively out of packet-interfaces, the UCP can send specially generated test-packets out of all known ports on switches that it has control over. When such a test-packet is received back by the UCP from a switch other than the one it was sent out of, the UCP can determine which port on the receiving switch is connected to which port on the sending switch (Fig. 2.14a).

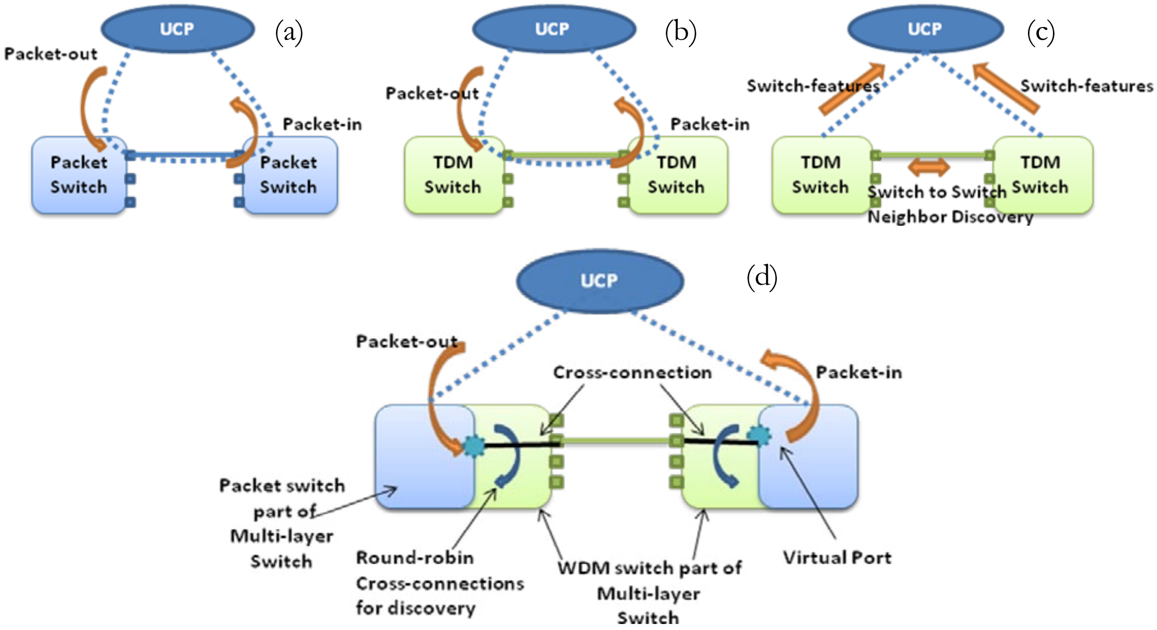


Figure 2.14: Discovery of packet and circuit links

TDM Circuit Links: On circuit switches such a mechanism is possible, but slightly more involved. Consider first a TDM switch based on SONET/SDH (but applicable to TDM switches based on OTN as well). SONET switches periodically send out SONET frames on all interfaces (eg. 8000 frames/sec). Each frame consists of a payload and a header, and there exists special header bytes (the DCC bytes [5]) reserved for packet-communications. These bytes can be used to carry the packets sent by the packet-out mechanism, with the understanding that software on the receiving end must put together the packet (spread over the DCC bytes in multiple consecutive frames), then add its receiving port and switch-id and make it available to the UCP. This scenario is depicted in Fig. 2.14b and is a viable way to support discovery of physical circuit-links.

Another way to achieve the same result for both packet and circuit *physical* links is to have the switches themselves perform this discovery using software/hardware *outside* the purview of the UCP. Since a lot of existing equipment already uses these DCC bytes to do neighbor discovery, all that is required is to report the *results* of the discovery process to the UCP as part of the switch-API (Fig. 2.14c).

One advantage of doing some link-layer tasks in the switches is that very fast keep-alives can be sent switch-to-switch to monitor the health of the link (link up-or-down) and save the UCP from processing the keep-alive. However, the disadvantage is that these protocols are vendor proprietary today making them hard to interoperate for discovery or keep-alives. We feel that both cases should be allowed, together with the use of standardized neighbor discovery (like LLDP [31]) and fault-detection protocols (like BFD [32]).

WDM Circuit Links: In the case of wavelength and fiber based circuit switches, where there is no visibility into packets, neighbor discovery and therefore link-discovery is hard to achieve. However we find that supporting the trend towards multi-layer switches, a lot of wavelength-switches are supporting packet interfaces with limited packet switching capability [33, 34]. We can then support the discovery of circuit links in wavelength/fiber switches via mechanisms shown in Fig. 2.14d. A packet sent via a packet-out message is periodically sent out of a virtual-port which is cross-connected to one of the interfaces. A virtual port on a receiving switch is cross-connected in round-robin order with all circuit ports. When one of the test-packets periodically sent out is eventually received successfully on a particular connection, the corresponding-link can be discovered.

However, this procedure has two drawbacks – a) it is time consuming as the round-robin nature needs to be repeated for all ports in all switches; b) it is disruptive to service as while this discovery procedure is going on, live traffic cannot be carried over the switches. For these two reasons, in this case, instead of doing live discovery it may be better to ‘configure’ the switches with their neighbor switch and port ids, so that the switch can report it to the UCP via the switch-API.

Given the above argument and keeping Fig.2.14c in mind, we included the ability to report peer switch-id and port-id per switch circuit-port in the *switch-API* as neighbor-discovery support; as well as the packet-in/out mechanism for UCP based discovery in the switch-API. Note that use of this API feature means that the UCP is not directly involved in the discovery process. Thus the reporting of neighbor information requires the UCP to have a verification methodology that ensures that different switches which report each other as peers on a certain port, report their peer’s switch and port id correctly. A circuit or packet link reported this way is deemed discovered only when both sides of the link report the other end correctly.

Constructing Layers: In discussing the common-map abstraction we said that we can choose whether to treat packet and circuit switches as part of the same topology or as part of different topologies (and thus different layers).

Creating network applications across packets and circuits is simple when we treat both kinds of switches as part of the same topology or layer (a sea-of-switches). On one hand, the right-side of Fig. 2.15 shows how *physical* packet links can be discovered by the controller (using test-packet-in and outs) and physical circuit links can be reported by the circuit (or multi-layer) switches and verified by the UCP, using any of the mechanisms shown in Fig. 2.14. This allows the creation of a single *physical* topology comprising of a sea of packet and circuit switches. We show an example of a network application on top of such a topology in Chapter 3.

On the other hand, the left-side of Fig. 2.14 shows a mechanism for creating separate packet and circuit topologies, where the network application treats them as separate *layers*. The packet layer (or topology) can consist of physical packet links when they link together packet switches that do not go over the wide-area. These links can be discovered by the packet-in-and-out mechanism provided by the switch-API. Similarly physical circuit links can be discovered or verified by the methods mentioned in the previous section. These links then form a separate circuit-topology (or layer). Importantly the circuit layer also includes the packet-switches that are at the border of the packet-network and connect physically to circuit-switches. These physical connections (either packet or circuit links) are also part of the circuit-topology as they contain vital information for mapping packet flows to circuit flows.

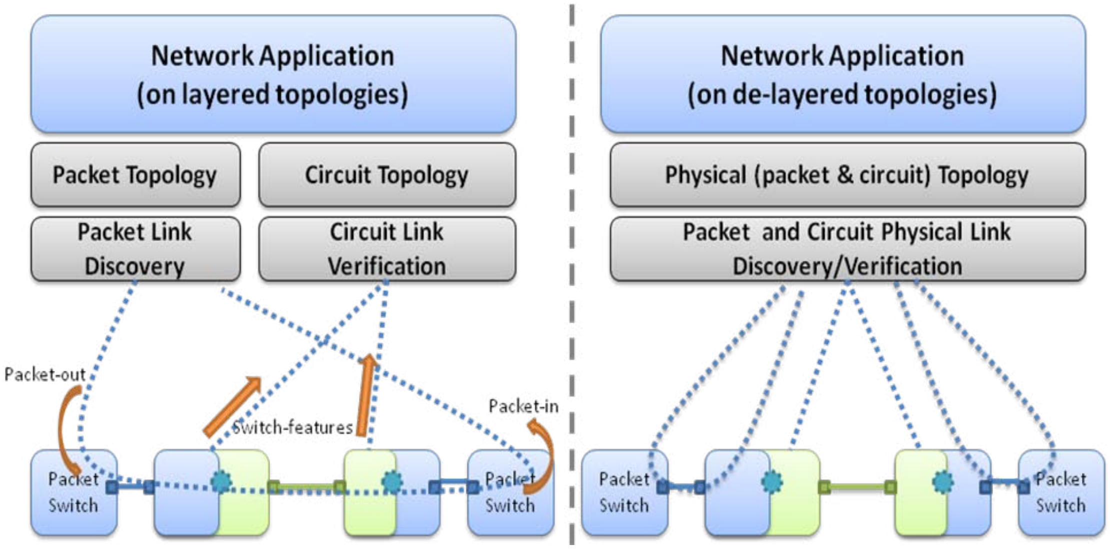


Figure 2.15: Layering and the Common-Map Abstraction

Discovery of virtual-packet links over the wide area can be performed by tunneling test-packets from packet switches over circuits created to support the virtual-packet-link. At the other end of the tunnel, the test-packets are received by the UCP completing the discovery of the virtual packet-link across the wide area.

Common-Map Maintenance: Entities such as ports and nodes report their status to the UCP via the switch-API. Additionally the status of links can be discovered, implied (from port status) and verified by the UCP via the switch-API. The common-map is updated and maintained by reacting to these status updates by creating/removing/or updating the relevant data-structures for the entities.

2.2.3 Network-API

A network-API for the common-map can have two parts: one that allows switch-level control and the other allows network-wide control. We consider them separately below. In discussing the network-API we do not distinguish between API calls for packet flows and circuit flows. The set of calls described, apply to both packet switched networks and circuit switched networks. The only difference is in the switching technology. Thus the implementation of these API calls is typically technology specific. For example, a function-call to setup a flow can involve installation of only packet-header fields in packet flow tables, while a circuit flow setup can include creating virtual ports at the end-points and setting up switch-fabric-specific (WDM, TDM) cross-connects for the circuit.

Switch-Level Control: A network application may require access to (internal) switch level detail for a variety of reasons depending on where the switch is located in the network topology and the kind of network function being targeted – examples include access-control, specialized/advanced forwarding (multipath-selection), monitoring queue depths etc. In such cases, the network-API essentially provides a wrapper around the switch-API calls. Thus all (or most of) the switch-API calls discussed in Secs. 2.1.1 and 2.1.2 could in principle be wrapped into a network-API call as (network-node-id, *switch-API*-call). Such calls include commands that the application can use to change/configure attributes in the switches and control the forwarding path within a switch. Also almost all network-wide monitoring is done on a per-switch basis. The application can register to be notified when such monitoring-events happen. Behind the scenes, the network API wrapper sets the switches using the 'trap' functionality of the switch-API.

Network-Level Control: Network applications may require the following min-set of capabilities from the network-API:

- Topology-choice: Depending on requirements the common-map abstraction may provide a single-topology or multiple topologies – two common examples are a) virtual-topology of packet-links on a physical-topology of circuit links, and b) a virtual topology of tunnels (eg. MPLS) on a physical-topology of packet-links. The application needs to choose which topology it works with for routing etc.

- Routing: In general routing can itself be considered as an application on top of the network-map. The user can define a routing-algorithm to support the following API calls, or the UCP could include a built-in generic routing-algorithm that supports the API. One example of a generic routing-API uses a Constrained Shortest Path First (CSPF) algorithm that finds the shortest path in the network given certain constraints. Such constraints can be bandwidth, delay, or any other user-defined link - attribute.
    - Methods to set attributes for links in the chosen topology before routing can be performed. Simple attributes are link-costs and maximum reservable bandwidth.
    - Method to get a route (get_route) on the chosen topology between source and destination nodes, depending on a specified set of constraints– this is essentially a check to see if a route can be found that meets all the constraints (applies well to MPLS-TE and circuit networks). Note that get_route can be run with multiple constraints simultaneously, or with no constraints at all. In the latter case the caller would get the path with the shortest cost. If all link costs are the same, the route would have the shortest number of hops.
    - Method to check an explicit route - the application can specify an explicit route by detailing the nodes along a path from source to destination. This call will verify the validity of the explicit route, given the current state of the network and the specified constraints.
    - Method to check an existing route – the application can specify an existing route explicitly and check for the feasibility of a change in a constraint along that route. For example, applications can check if a flow (with reserved-bandwidth) can increase its bandwidth-reservation along the path it currently takes.
    - Method to detail a loose-route – loose routes are partially specified routes. This call fills in the details of the nodes along the loose route by checking for all the specified constraints along the loose route.

- Flow-setup:
    - Method to setup a flow (packet or circuit flow) in the network – given a route, and a set of <flow_definitions, associated actions> for each node along the route, a flow is created (packet-flow, circuit –flow, virtual-circuit-flow). Under the hood, the UCP sets up the flow-table entries in each switch along the route using the switch-API. If such a call changes the link-attributes then the common-map is updated. For example when a circuit is created with a certain bandwidth, it consumes that bandwidth along the links the route traverses. Thus the links in the common-map have their attributes updated.
    - Method to delete a flow (which updates the map if necessary)
    - Method to re-route a flow by changing the set of actions associate with the flow or changing the route.
    - Method by which an application can register for a type-of flow so that if matching packets arrive then they can be routed.
- Recovery:
    - Methods to inform applications of changes in network topology (link or node failures) so that the application can figure out new routes and install them.
    - Method to configure the routing engine to automatically respond to change in network topology by re-routing flows without waiting for the application to explicitly re-route them.
    - Methods to set advanced re-routing state, so that the switches have pre-installed state for re-routing when network failures occur. Such pre-installed state could be backup paths (like MPLS Fast Reroute or circuit protection paths)
- Network-wide Monitoring:
    - Methods for the application to be notified of network-state along a route or on specific links eg. congestion
    - Methods for sampling packets along installed packet-flows in the network.

2.3 pac.c Prototypes

We discuss three prototypes we built to validate our architectural and control plane constructs [35]. The Unified Control Plane (UCP) from Fig. 2.16 consists of:

- OpenFlow: An interface/protocol that instantiates the common-flow abstraction by enabling the switch-API into packet and circuit switches. Our work extended version 1.0 of the protocol [28, 36] for circuit switches (Sec. 2.1.2).
- A Controller running a network-wide Operating System called NOX [15]. We instantiated our common-map abstraction by building and maintaining the common-map and network API on top of NOX, with ideas discussed in Sec 2.2
- A Slicing Plane (not implemented) which is crucial to the practical deployment of the common-map, as we show in Chapter 3.

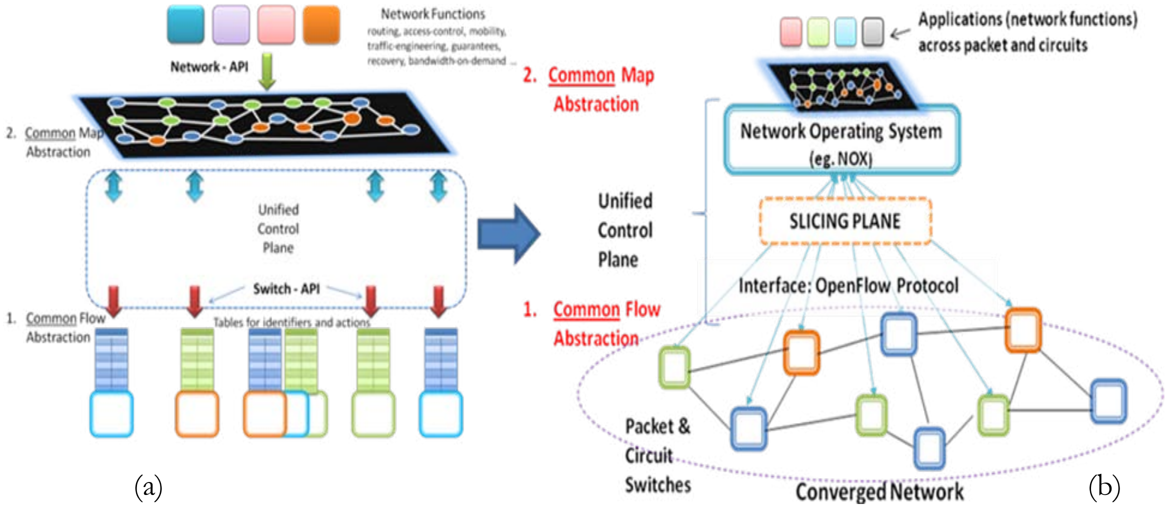

Network Functions
routing, access-control, mobility, traffic-engineering, guarantees, recovery, bandwidth-on-demand ...
Network - API
2. Common Map Abstraction
Unified Control Plane
Switch - API
1. Common Flow Abstraction
Tables for identifiers and actions
(a)
Applications (network functions) across packet and circuits
2. Common Map Abstraction
Network Operating System (eg. NOX)
Unified Control Plane
SLICING PLANE
Interface: OpenFlow Protocol
1. Common Flow Abstraction
Packet & Circuit Switches
Converged Network
(b)

Figure 2.16: Implementation of our Architectural Approach

We first present two early proof-of-concept prototypes we built with two different kinds of circuit switches - wavelength & TDM. These prototypes focused on validating the flow-abstraction. Our third and more complete pac.c prototype validates the common-map abstraction.

2.3.1 Prototype #1: Packet & Wavelength Switch Control

Goal: Validation of the common-flow abstraction by demonstration [37, 38] of unified control over OpenFlow enabled packet switches and an OpenFlow enabled optical wavelength switch.

Data-Plane: The main circuit switching element is a Wavelength Selective Switch (WSS) from Fujitsu which forms the basis of the Flashwave ROADM systems [34]). The WSS is an all-optical wavelength- switch in a 1X9 configuration. It has the ability to independently switch any of 40 incoming wavelengths at the single input port, to any of 9 output ports (Fig. 2.17). The incoming wavelengths (100 GHz spaced, ITU C-band) are de-multiplexed and directed to their respective MEMS mirrors, which are rotated to direct the wavelength to any of the 9 output ports, where they are multiplexed back into the outgoing fiber.

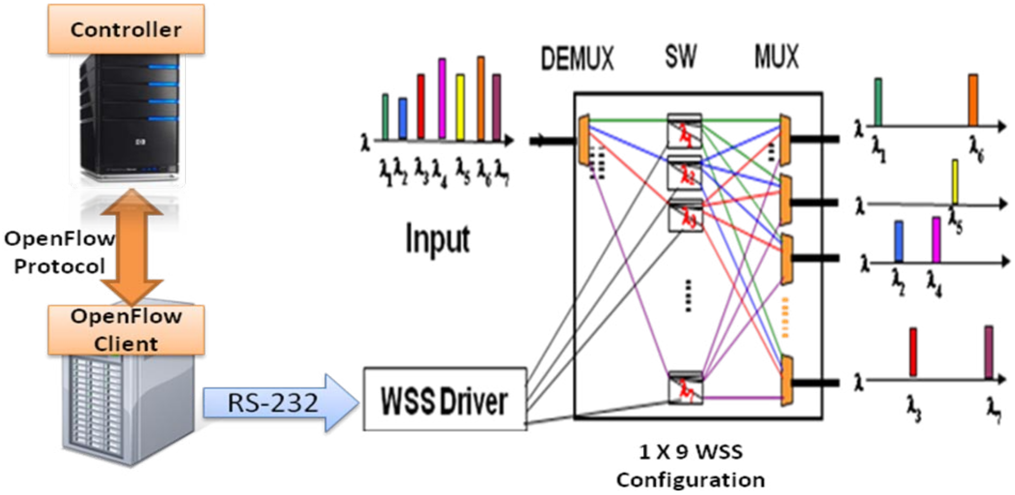


Figure 2.17: OpenFlow enabled Wavelength Selective Switch(WSS)

The WSS mirrors are controlled with a voltage-driver, which is sent commands over RS232 from a PC. The PC runs a modified version of the OpenFlow reference switch [39]. We used the OpenFlow client part of the code which interacts with the Controller and modified according to our changes to the OpenFlow protocol [36]. The rest of the code that implements packet-switching in software is eliminated. The modified client was integrated with an RS-232 driver to send commands to the voltage-driver that directs the mirrors. Together the OpenFlow-client PC, voltage-driver and the WSS can be regarded as an OpenFlow-enabled circuit switch.

The data plane also consists of two OpenFlow enabled packet-switches implemented in software with the OpenFlow reference switch [39]. The switches can be hosted in any PC with multiple Ethernet interfaces. In our testbed we used PCs with NetFPGAs[*] as 4-port Network Interface Cards (NIC) [40].

Control-Plane: Our controller was implemented by making changes to the simple reference-controller that is distributed with the reference switch [39]. The focus in this prototype was on implementing the common-flow abstraction correctly. As a result the simple controller used in this prototype does not really create the common-map. Instead it treats each packet-switch as independent Ethernet learning-switches.

Experimental Setup: Fig. 2.18 shows our experimental setup. Two packet-switches NF1 and NF2 are connected via an optical link. Each packet-switch has four Gigabit-Ethernet (GE) electrical interfaces. One of the electrical ports (on each switch) was converted to an optical interface via a GE-to-SFP electrical-to-optical converter from TrendNet [43]. We used 2.5 Gbps SFP transceiver modules from Fujitsu in the converter, which transmitted DWDM ITU grid wavelengths 553.3 nm and 1554.1 nm. These wavelengths travel in opposite directions to form the bidirectional optical link.

The optical link comprised of 25km of single-mode fiber separating the two packet switches. The two wavelengths that form the bidirectional link were multiplexed/de-multiplexed into the fiber at the output/input of NF1 by a wavelength mux/demux (AWG). At the other end of the link, the wavelengths are again multiplexed/de-multiplexed into NF2 by the wavelength switch (as the switch has mux/demux capabilities as well). Initially however the wavelength *switch* is ‘open’ and so the wavelength coming out of NF2’s transmitter (1554.1nm) does not reach NF1, and the wavelength transmitted by NF1 (1553.3nm) does not reach NF2. In other words the

[*] The NetFPGA is a programmable hardware platform [41]. Implementations are available now that can be installed in the NetFPGA to make it behave like an OpenFlow enabled hardware packet-switch [42].

optical- link is broken, and so is the Ethernet link it supports between NF1 and NF2. The optical link is monitored by tapping-off a small percentage of the light (2%) before the receivers in both directions, and feeding the tapped-light into an Optical Spectrum Analyzer (OSA). Finally we connected client PCs (end-hosts) to the other GE interfaces on NF1 and a video-server PC to NF2.

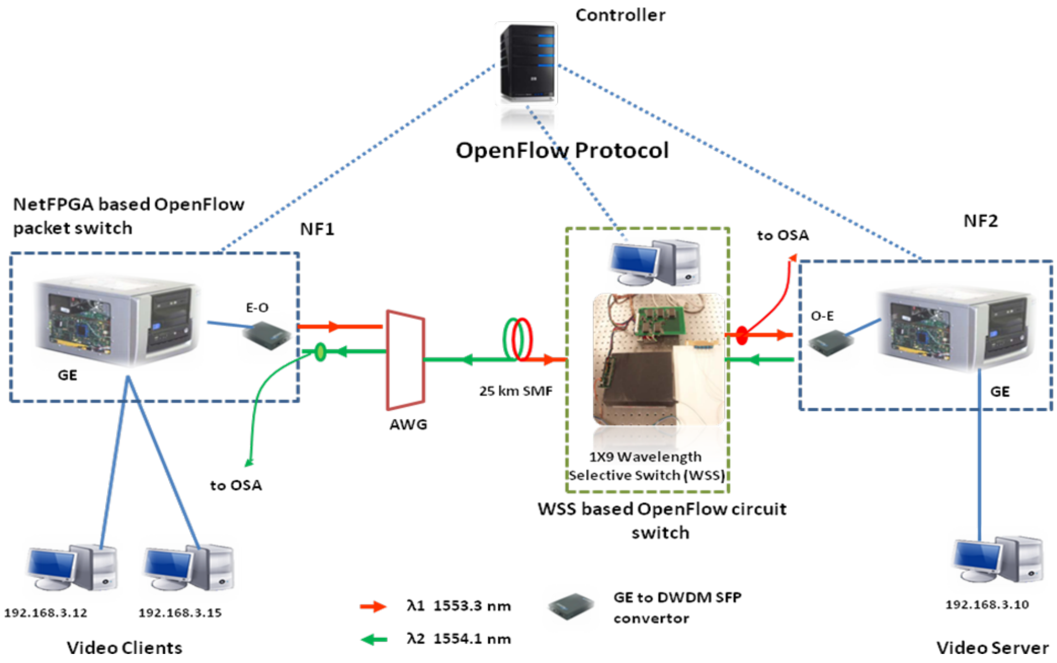


Figure 2.18: Prototype #1 - Packet and Wavelength switches

Experiments & Results: The basic idea is that the video-clients make requests for videos from the video-server. The video request is transported over TCP. But initially the client PC is not aware of the MAC address corresponding to the IP address of the video-server. It therefore sends out an ARP request, which is received by NF1. Since the ARP packet does not match any flow entries in NF1[†], it gets forwarded to the OpenFlow controller as a packet-in message.

Upon receiving the ARP packet, the controller decides that in order to reach the server, a circuit needs to be created between NF1 and NF2. The controller uses the OpenFlow protocol to insert rules into the WSS cross-connect table, thereby making the

[†] Actually all flow-tables (in NF1, NF2 and the WSS) are empty at the start of the experiment.

cross-connection for each wavelength in the WSS. Once the bi-directional optical link is up, the Ethernet link also comes up between NF1 and NF2. The controller also inserts flow table entries in the packet-switches to broadcast the ARP request to all interfaces other than the one in which it received the packet. This results in the ARP request reaching the server PC via the WSS and NF2. The server PC sends the ARP reply, following which TCP handshaking takes place and the video request is transported to the server. To serve the video request, the server streams the video data packets using RTP over UDP. These packets are transported over the same bidirectional circuit created by the controller, and are received and displayed by the video-client.

We measured the time taken by various steps of the process (Table 2.1). The measurements were made using Wireshark running on NF1, NF2 and the WSS driver PC. Details of the configurations and the measurement procedures can be found in [37].

Steps	GE-Optical-GE link with WSS initially not cross-connected	GE-Optical-GE link with WSS already cross-connected	GE-GE link (no optics)
WSS connect command received	1 ms	-	-
Cross-connection confirmation	1.3 s	-	-
TrendNet GE-SFP module link-up	4.74 s	4.10 s	-
NetFPGA GE-GE link-up	1 ms	1 ms	1 ms

Table 2.1: Prototype #1: Time taken for connection set up

Conclusions: We make the following observations from our first prototype:

- The most important lesson from this prototype is the feasibility of our common-flow abstraction; that it was possible to treat OpenFlow-enabled packet and circuit switches as flow-tables that switch at different granularities (packet and lambda) and can be commonly-controlled by an external controller using a common switch-API.
- We found that the time taken to create a wavelength cross-connection is high (1.3sec). However this has nothing to do with our implementation of the OpenFlow protocol.

Instead the time is actually taken by the RS-232 communication between the driver PC and the WSS voltage-driver, which is not optimized for fast setup of connections. For example, while the mirrors can be rotated simultaneously, the cross-connect commands (over RS-232) are sent one at a time, and the second command cannot be sent until detailed feedback is read off from the serial port for the first command. We believe that optimizing this interface can reduce this number to tens of ms.

- The time taken by the GE-SFP convertor to recognize a link-up is high (4.10s), most likely because the internal mechanisms of the converter box have not been optimized for rapid link-up. We believe that a dedicated ASIC/FPGA optimized for fast setup will help in this regard.

2.3.2 Prototype #2: VLAN & TDM Switch Control

Goals: To build on our implementation of the common-flow abstraction by abstracting a different kind of circuit switch – a SONET/SDH based TDM switch; To explore the implementation of a simple-application across packet-and-circuit switches.

Data-Plane: The data-plane consists of carrier-class Ciena CoreDirector (CD/CI) switches [44].The CDs are hybrid switches – they have both Layer 2 (GE) interfaces with a packet switching fabric, as well as Layer 1 (SONET/SDH) interfaces with a TDM switching fabric (Fig. 2.19a). The packet switching capabilities on the CD are limited to switching on the basis of VLANs (and incoming- port). Nevertheless, the prototype has both packet and circuit switches (albeit housed in the same box).

Working with Ciena's development team, the OpenFlow protocol was extended to serve as a switch-API into the CD's VLAN and TDM (SONET) switching fabrics. Ciena's development team added native support in their switches for the OpenFlow protocol (and its circuit-extensions [36]) to serve as a switch-API. Simultaneously, we worked on a controller to communicate with the CD and run unit-tests to debug the development of the OpenFlow client in the CD.

Control Plane: The control plane featured a controller running NOX [15] over an out-of-band Ethernet network. NOX was modified to include our changes to the OpenFlow protocol for circuit switching. We retained only the basic NOX event-engine and connection-handler part of NOX (known as nox-core – see Fig. 2.19b). This early prototype did not include any of the discovery and layering features presented in Section 2.2. Neither did it use the built in features in NOX for packet link discovery and topology, as there aren't any standalone packet switches in this prototype (like the software packet-switches in Prototype #1). The map abstraction for this topology only applies to the circuit-topology which was statically defined (i.e. the topology was hard-coded; not discovered). The circuit-API shown in NOX was basically a wrapper around the switch-API commands in OpenFlow. The controller also interfaces with a GUI that shows network state in real-time (Fig. 2.20b, created using ENVI [47]).

Experimental Setup: We used three CDs in our prototype connected to each other via OC-48 (2.5 Gbps) SONET/SDH links (Fig. 2.20a). Similar to prototype #1, we connected video clients and server PCs to our switches, except this time instead of standalone software packet switches, we connected the PCs directly to the Ethernet (GE) interfaces on the CDs. The three CDs together form a small demo-network.

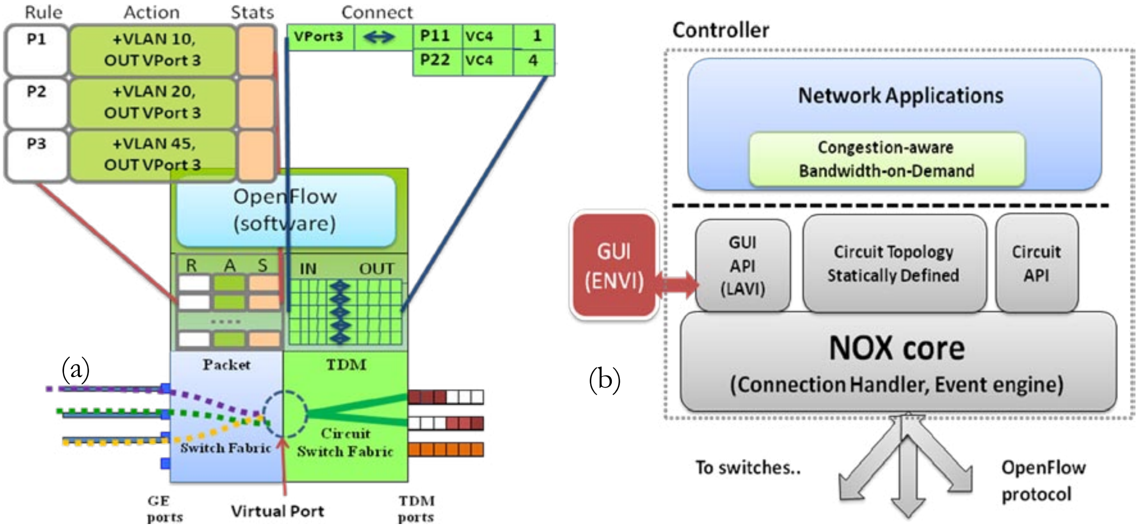


Figure 2.19: (a) CD internals and Flow Tables (b) Controller internals

Experiments: The focus was still on validating the switch-API (OpenFlow) for TDM switching, and developing the common-flow abstraction by interfacing TDM flows with VLAN based packet-flows. The best way to show this was by demonstrating the capability in NOX to set-up, modify and tear-down both L1 (SONET) and L2 (VLAN) flows on–demand. Once we had the ability to dynamically create and map packet-flows to circuit-flows, we wished to explore a simple network-application across them. And so, we created an application that dynamically responds to and relieves network congestion by adding bandwidth on-demand (Fig. 2.16b) [45, 46]. We discuss each of these experiments next.

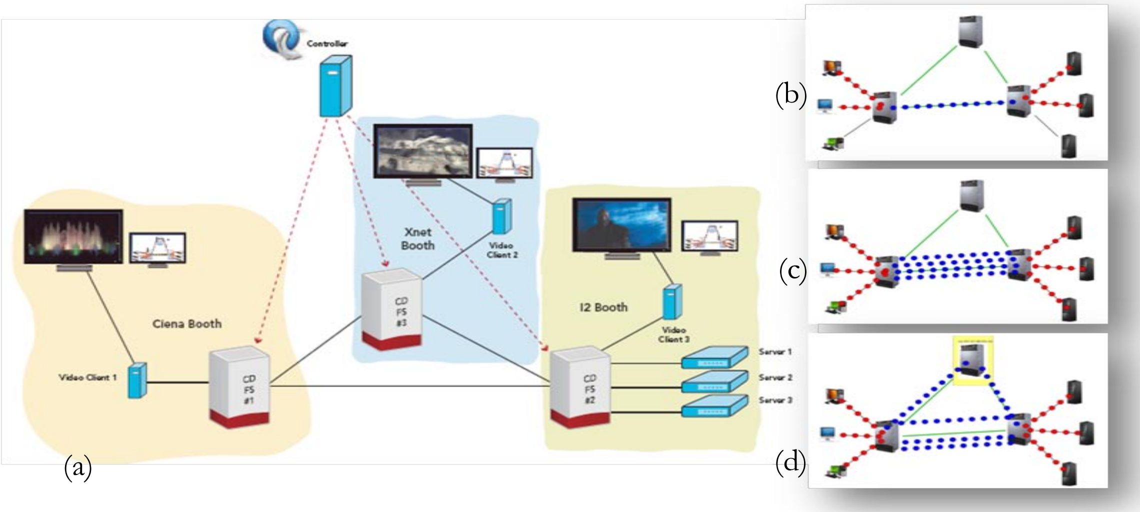


Figure 2.20: Prototype #2 Experimental Setup and Demo GUI

<u>L1 and L2 control</u>: The objective is similar to the one in the previous prototype where video-clients make requests for videos from remote streaming-video servers. When such packets (TCP SYN) from the first client arrive at the GE interfaces of the CDs, the CD tries to match the packet to rules in its packet-flow-table. Since the flow-table does not have entries for such packets, they are redirected to the OpenFlow controller (as packet-ins). The controller decides that for the packets to reach the IP address of the server, there needs to be a circuit-flow between CDs #1 and #2. Thus our application:

- Creates virtual-ports in the CDs to serve as end-points for circuit-flows, where packet-flows can be mapped into them – eg. VPort3 in Fig, 2.19a ;
- Inserts rules in the packet-flow tables that dictate that all packets coming in from client GE-port (eg. P1 in CD #1) and the video server GE-port (in CD #2) are tagged with a particular VLAN-id (eg. VLAN10 for P1 in CD#1) and forwarded to the virtual ports in the respective CDs (as shown in Fig. 2.19a);
- Inserts rules in the packet-flow tables for the *opposite* direction, which stipulate that all packets matching the <virtual port and VLAN-id> combination, have the VLAN tag stripped and then forwarded to the GE port corresponding to the client-PC (not shown in Fig. 2.19a);
- Inserts rules in the cross-connect tables that connect the virtual port to SONET signals (timeslots) that connect the two CDs bi-directionally (eg. in Fig. 2.19a, VPort3 is cross-connected to two VC4s – 150Mbps each – which start on timeslots 1 and 4 on ports P11 and P22 respectively). The cumulative circuit-bandwidth is then 300Mbps[^].

All subsequent packets (in both directions) for this client-server pair match the existing flow definitions and get directly forwarded in hardware. For other client-server pairs the application chooses a different VLAN tag, so the CD's packet-switch fabric can distinguish between packets coming in from the virtual-port and destined to different client/server pairs. As video data is received from the server, the packets are tagged with the internal VLAN id and mapped to the virtual-port. At the client side, the packets received from the virtual-port are switched to the client port based on the VLAN tag, which is then stripped off before the packets are forwarded to the client PCs, where the video is displayed on the screen. Packet flows are shown in the GUI (Fig. 2.20b) between the PCs and the CDs, and circuit flows are shown between the CDs.

Congestion-aware Bandwidth-on-Demand: Initially, the cumulative data-rate of two video streams is less that the bandwidth of the circuit flow they are multiplexed into (Fig. 2.20b), and the videos play smoothly on the client PC displays. However, when a third

[^] The virtual-port is created using SONET Virtual Concatenation (VCAT) and Link Capacity Adjustment Scheme (LCAS) features [5].

video stream is multiplexed into the same circuit-flow, the bandwidth is exceeded, packets start getting dropped, congestion develops in the network, and the video displays stall. However our application monitors network performance by acquiring circuit-flow statistics from the virtual ports in the CDs. It becomes aware of the packet drops, makes sure that the congestion is due to long-lived flows, and then responds by increasing the circuit bandwidth by adding more TDM signals to the virtual-port (Fig. 2.20c); thereby relieving congestion & restoring the video streams which start displaying smoothly again.

Conclusions: From this second prototype, we arrived at the following conclusions:

- Validation of our common-flow abstraction; that it was possible to treat a hybrid packet-circuit switch as a combination of flow-tables that switch at different granularities (packet and time-slot) and can be commonly-controlled by an external controller using a common switch-API.
- Validation of the flexibility of virtual-ports as the mapping-functions between packet and circuit domains. Prototype #1 had the virtual-port represented by the physical GE-wavelength (SFP) convertor-ports in the packet switches, and Prototype #2 had the virtual port represented by a GE-TDM mapper in the hybrid switches.
- We learnt a number of lessons about how the API can be improved. For example, our TDM port bandwidth representation in v0.2 of the extensions (which we used in this prototype [35]) is cumbersome - and so in v0.3, we use a much simpler representation [36]. Another example relates to the use of a common structure to represent packet & circuit ports. In v3 we use separate structures, as it helps implementation. For example, vendors of packet-only-switches need not implement circuit-related features of the OpenFlow protocol (switch-API) and vice-versa.
- With the common-flow abstraction feasibly instantiated, we made our first foray into developing network-applications that worked across packets and circuits. But we did so without the important features of discovery and fault-detection, which help create the common-map and keep it updated. This was fixed in the next prototype.

2.3.3 Prototype #3: Full pac.c Prototype

The early prototypes discussed in the previous sections were useful as proof-of-concept demonstrations of our architectural constructs. They were also useful for demonstrating simple applications in the controller across packets and circuits. However they were limited in the following ways – the prototype in Sec. 2.3.1 had standalone packet switches and a wavelength switch, but it was only a single-link demonstration; the prototype in Section 2.3.2 was more involved but had limited packet switching capability (only VLAN); finally both prototypes lacked a more evolved implementation of the common-map abstraction as described in Sec. 2.2 (no discovery, recovery etc.) In this section and the next Chapter, we describe our evolved and complete pac.c prototype.

Goals: The initial goal of this prototype was to instantiate the common-map abstraction by creating and maintaining a common-map and network-API. Then building on the initial goal: we wished to create a full pac.c prototype network with standalone packet and circuit switches that emulated WAN *structure*; we wished to provide the choice of treating packet and circuits in different layers; and use different data-plane packet-flow to circuit-flow mappings than what had been previously demonstrated. Finally, the ultimate goal of this prototype was to enable the creation of a fairly involved network application across packets and circuits; and then using this experience to validate our architectural claims of simplicity and extensibility.

Data-Plane: In part, the data-plane consists of the same three Ciena CoreDirector (CD) hybrid packet-circuit switches from the previous prototype (Fig, 2.21). The OpenFlow client firmware inside the CDs was upgraded to support all the features of version 0.3 of the circuit switching extensions to the OpenFlow protocol [36]. Discovery and recovery features were added and other deficiencies (mentioned in previous section's conclusions) were corrected in this version of the extensions. The CD firmware was also upgraded to support version1.0 of the packet-switching part of the OpenFlow protocol (previous prototype was based on v0.8.9).

In addition to the hybrid switches, in this prototype we added standalone packet-switches with full capability to support packet-switching based on the OpenFlow spec v1.0 [28] (Fig.2.21). We used a single 48 port GE switch from Pronto (earlier Quanta [48]) supporting the Indigo firmware [49]. The single switch was 'sliced' to behave like *seven independent* packet-switches each with 6 GE ports. The slicing plane was based on a modified version of the FlowVisor [50]. The basic idea of the FlowVisior is that a switch can be sliced to behave like multiple switches with each slice of the switch under the control of a different controller. But if we give the control of each slice to the same controller, then the sliced switch appears to that controller as multiple switches.

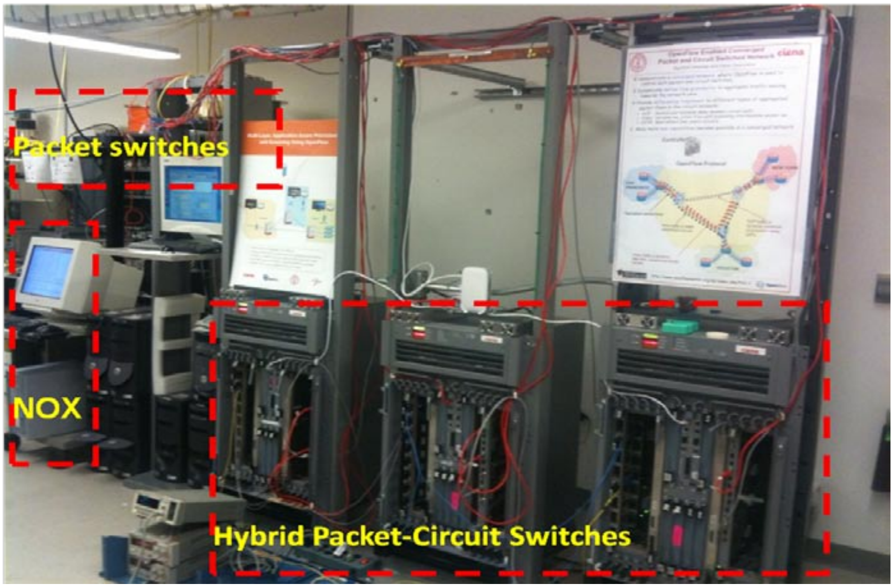


Figure 2.21: Full pac.c Prototype

Control-Plane: In this prototype we focused initially on developing the common-map abstraction. Similar to Prototype #2 we again used NOX as our controller. NOX was written mainly for enterprise networks, which have host-machines and middleware directly connected to the network[†]. In our work, to develop the common-map abstraction for carrier networks, we ignored most of the LAN-related-functionality that NOX provides (eg. authenticator, host-namespace, DHCP/DNS modules etc.).

[†] The NOX paper [15] briefly discusses the internal architecture of the network-OS.

The parts of NOX we did use are the basic event-engine and connection handlers that are collectively referred to as nox-core (similar to Prototype #2). Additionally some modules we used, that come built-in [51], include packet-link discovery and packet-topology, together with library-support and GUI API's (Fig. 2.22). Network applications (shown above the horizontal dashed-line) such as routing, can use these modules to implement network-functions; together the built-in modules create a network map for packet networks and keep it updated with network state.

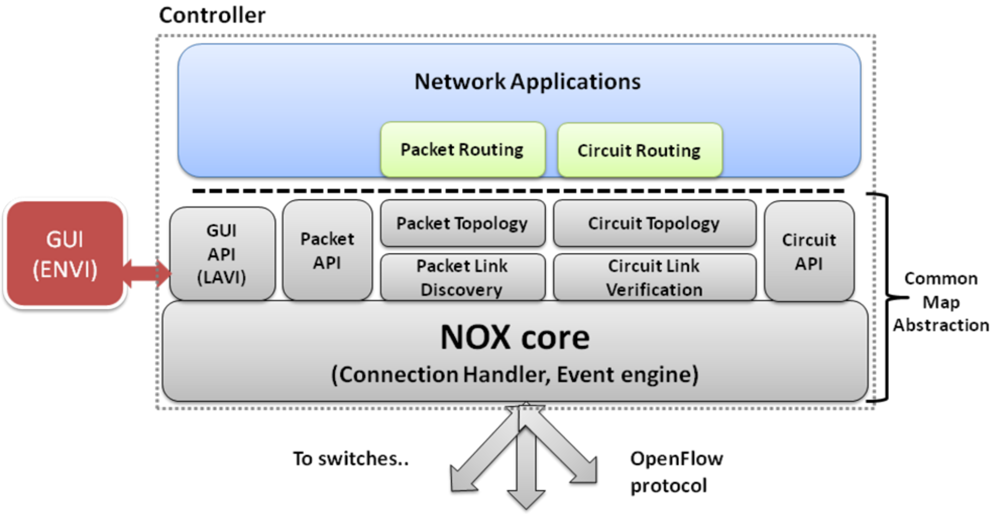


Figure 2.22: Common-Map Abstraction instantiated in NOX

In NOX-core we changed the definition file for the OpenFlow protocol, in order to include the changes we made to the protocol to support the common-flow abstraction (Sec. 2.1). The switch-features message allows us to build up the map representation for nodes, ports, flow-tables, and queues. We discover physical packet-links via the NOX's discovery module. For reasons mentioned in Sec. 2.2.1, we have the TDM switches perform neighbor-discovery and report on their peers as part of switch-features. We added a circuit-link verification module (Fig. 2.22) to ensure that the peers reported by the switches on either end of a circuit-link matched-up, and only then were the links included in the topology.

At this point, both physical packet and circuit links had been discovered. However we wished to enable the treatment of packet and circuit switches as part of different *layers* (as discussed with respect to Fig. 2.15). The reason relates to our ultimate goal of using this prototype to validate our architectural claims. In order to do that we need to compare our work with the way industry 'sees' these networks today – as part of different layers. And so we followed the procedure briefly mentioned in Sec. 2.2.2 to create these layers with separate topologies. Importantly, since we already had all physical links, we needed a way to discover virtual-packet-links, which are supported by underlying circuits over the wide-area.

We detail the procedure followed in the context of NOX. We quote from [15] (with small clarifications): "To cope with network change events, NOX applications use a set of (event) handlers that are registered to execute whenever a particular event happens. Event handlers are executed in order of their *priority* (which is specified during event registration). A handlers' return value indicates to NOX whether to stop execution of this event, or to continue by passing the event along to the next registered handler."

Using NOX's event priorities, discovery of virtual packet-links can be performed as follows: Consider the situation where two standalone packet switches are connected over the wide area through multi-layer transport switches like the CDs.

- Since there are two discovery modules in NOX, we give higher priority to the circuit-discovery module to received discovery events. The module can ignore or pass-on the discovery packet to the packet-discovery module (depending on the layering choice).
- The discovery-packet sent out through a standalone packet-switch is reported back to NOX by the packet part of the CD. The circuit discovery module receives it and discovers the physical packet link connecting the standalone packet-switch to the CD. But now it chooses to stop the execution of the event. So the packet-discovery module never learns about this physical packet-link.
- The circuit discovery module is already aware of the circuit-links via the CDs SONET discovery mechanisms. It then has all the information it needs to complete

the circuit topology – the CDs, the circuit links that connect them, the packet switches that are physically connected to the CDs and the physical-packet links being used for those connections.

- The controller then enters rules in the packet-switch part of the CD to tunnel all subsequent discovery test-packets from the packet-switch through the circuit-flow to the packet-switch at the other end of the circuit flow.
- At the other end of the circuit-flow, the discovery packets are reported back to the NOX by the other packet-switch; this time the test packet is *ignored* and passed on by the circuit-link discovery module; and received by the packet discovery module. The latter can therefore receive this message and determine the (virtual) packet-link, which inter-connects the standalone packet-switch interfaces via the circuit-flow over the wide area.

In this way the packet, the packet topology (i.e the packet-layer) can be completed with standalone packet switches and physical as well as virtual packet links, and kept separate from the circuit-layer (which includes the CDs and circuit-links).

We kept the common-map updated via two different mechanisms. The packet-layer uses the packet-discovery process as keep-alives to discover link-downs. The circuit-layer uses SONET's ability to quickly discovery link-down and report it to the controller as a port-status message; at which point the circuit-link goes down in the circuit topology, and the virtual-link it supports in the packet-topology may also go down (depending on the type of recovery mechanism selected[†]). Finally we also implemented the beginnings of a network-API (including routing and recovery from Sec. 2.2.3) to manipulate the common-map.

Experimental Setup: With this prototype we had most of the elements necessary to *emulate* a wide-area network. Importantly, we emulated WAN *structure*[*] i.e. packet-switches (access and core routers) clustered in a city's Point-of-Presence (PoP); with the core routers connected over the wide area (inter-city) by optical transport equipment.

[†] For example if all recovery is deemed to be handled by the circuit-layer, then the virtual-packet-link does not go down. It simply gets supported by a different recovery-circuit.

[*] as opposed to WAN latencies that come from propagation over thousands of miles in the wide-area. All our switches are housed in the same lab with only a few meters of fiber connecting them.

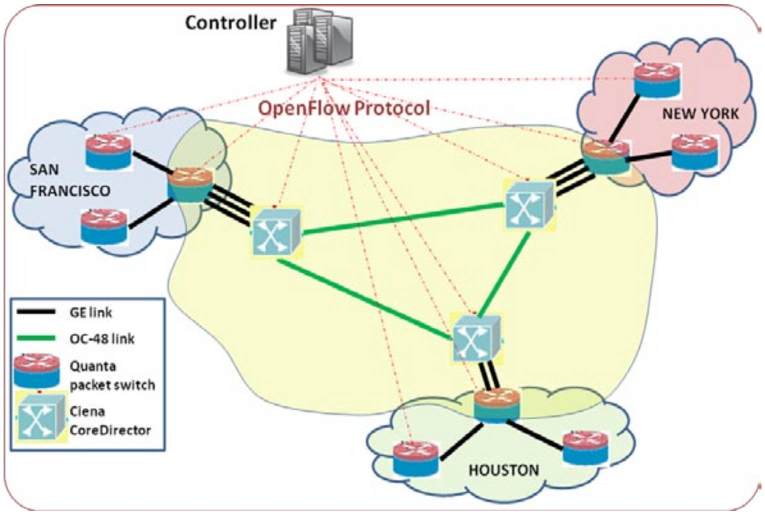


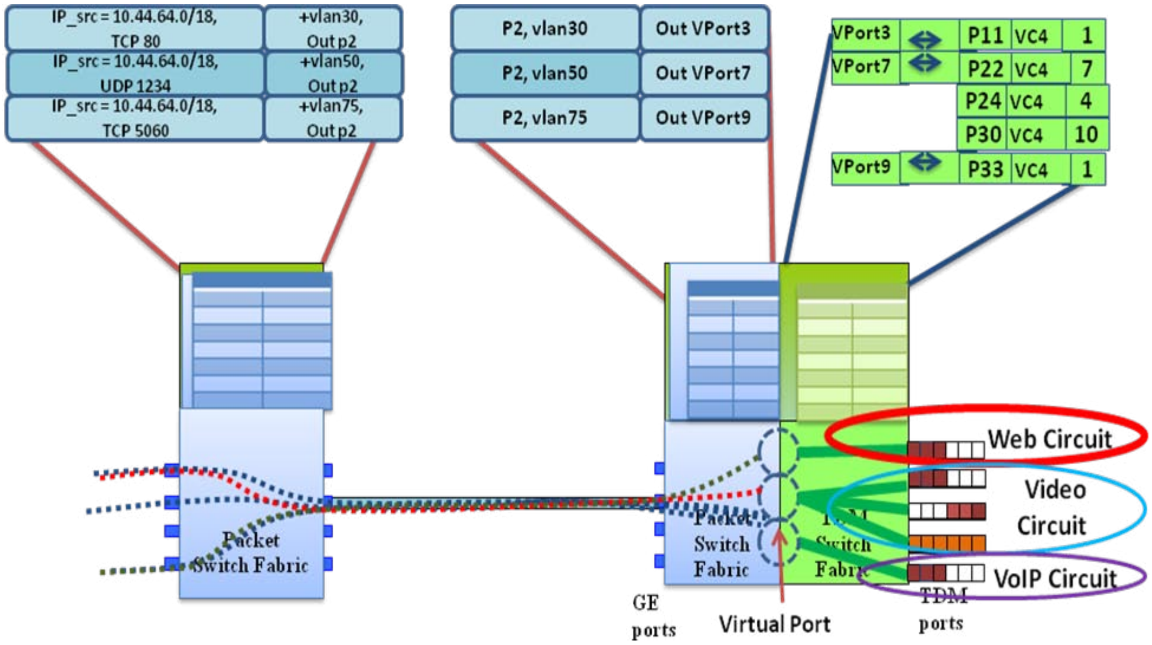
Figure 2.23: Prototype #3: WAN Emulation

The links shown *within* the cities in Fig. 2.23 correspond to physical GE links between the independent Quanta packet switches; and also between the packet switches and the CoreDirector (CD)'s Ethernet ports. For example, in the San Francisco (SFO) cloud, we see two (access) switches connected by GE links to a single core switch. The SFO core switch in turn is connected to the SFO CD via GE links. The CDs in turn are connected by OC-48 optical-fiber-links over the wide-area. As before, all the switches are OpenFlow enabled and communicate with the controller over an out-of-band Ethernet network (non-OpenFlow controlled).

Experiment: We have implemented a fairly complex network-function in a network-application on our controller, that makes use of both packets and circuits at the same time, to achieve the functionality. We defer the discussion of the entire application, and the conclusions we drew from the implementation, to the next chapter. Here we demonstrate some of the underlying flow-table manipulations our application drives with special emphasis on how packet-flows get mapped into different circuit flows. With this experiment we show how we can write an application that simultaneously controls flows on the basis of identifiers that belong to Layer 4 (TCP/UDP), L3 (IP), L2 (VLAN) and L1 (SONET).

Fig. 2.24 presents a snapshot of the forwarding tables in the SFO core packet-switch and the SFO transport-switch (CD) physically connected to it. The objective of this experiment is to show that packet-flows for different *types* of traffic (voice, video and web) can be identified and *mapped* into different circuit-flows.

In this snapshot, rules have been inserted into the SFO core packet-switch's flow-table that differentiate between three different types of traffic from the same customer – in other words, three different packet-flows have been created. The customer is identified by the IP source-address range of 10.44.64.0/18. The different traffic-types are differentiated on the basis of well-known (eg, HTTP traffic on TCP port 80) or registered (eg. VLC media player RTP/UDP stream) transport port numbers [55]. Thus each rule is a combination of the IP src-address (L3) and a different TCP/UDP destination port (L4).



IP_src = 10.44.64.0/18, TCP 80	+vlan30, Out p2
IP_src = 10.44.64.0/18, UDP 1234	+vlan50, Out p2
IP_src = 10.44.64.0/18, TCP 5060	+vlan75, Out p2

P2, vlan30	Out VPort3
P2, vlan50	Out VPort7
P2, vlan75	Out VPort9

VPort3	⟺	P11	VC4	1
VPort7	⟺	P22	VC4	7
		P24	VC4	4
		P30	VC4	10
VPort9	⟺	P33	VC4	1

Figure 2.24: Programming the Flow-Tables

The basic idea is to map the three different packet-flows into three independent circuit-flows. One way to achieve this could be to use three different physical-ports between the core-packet switch and the CD. But this is an expensive approach, especially since the application learns from the common-map that the transport switch's (the CDs) packet-forwarding table is capable of matching on <input-port, VLAN tags>; and is therefore capable of demultiplexing packet-flows coming in from the *same* physical port on the basis of different VLAN-ids. Thus the application performs the following actions on all packets that match the rules in the core-packet-switch. It adds VLAN tags (L2) with different VLAN ids (30, 50 and 75) and forwards the flows out of a *single* port (port#2) on the core-packet-switch[†].

The application also adds rules in the CD's packet-flow-table to match on incoming-port P2 and the different VLAN ids, with the action defined to forward all packets that match the rules to different *virtual ports* (end-points of circuit flows). For example, HTTP packets which had VLAN30 inserted into them by the core-packet switch, match the rule in the CD with the corresponding action that forwards them out of virtual-port 3. The virtual ports have been created beforehand by the application and they contain the mapping actions necessary to map Ethernet frames to SONET frames.

Finally the snapshot shows that the application has inserted rules that cross-connect the virtual-ports to time-slots on the physical SONET ports. For example, HTTP traffic (VPort3) is assigned 150 Mbps of bandwidth (VC4) on SONET port P11 (starting at time-slot1), while the video-traffic aggregate (VPort 7) gets 450 Mbps bandwidth (three VC4s) spread over 3 different SONET ports (using VCAT).

In this way packet-flows for three different traffic types from the same customer have been mapped into separate circuit-flows with different bandwidths. The application has the power to dynamically change any of these mappings as it sees fit. For example:

- The packet-flows could be bundled into the same circuit-flow by changing the VLAN ids in the core-packet-switch. Or by forwarding to the same virtual-port in the CD.
- Packet-flows from different customers can be mapped into the same circuit-flow by matching on the different customer IP-src address but adding the same VLAN tag.

[†] Note that if the packet-flow-table in the CD had full matching capabilities, we wouldn't need the VLAN tag insertion. The same flow-table rules based on IPsrc/TCP/UDP could have been used in the transport switch as well. However the use of a tag/label has another advantage. In a flow-table implemented with TCAMs, rules based just on tags or labels take less space.

- Bandwidth allocations for the circuit-flows can be changed on the fly by cross-connecting the virtual-ports to more (or less) time-slots. Time-slots can be re-assigned by cross-connecting them to other virtual ports.
- New circuit-flows can be created by adding new virtual-ports, cross-connecting them to time-slots, and re-directing packet-flows by forwarding them to the newly created virtual port

Conclusions: To summarize, we draw the following conclusions:

- We achieved our initial goal of instantiating the common-map abstraction, by using features built-in to NOX for packet-switching, and by adding a set of modules for discovery, topology and recovery for circuit-switching. Compare Fig. 2.22 to the usage of NOX in Prototype #2 (Fig. 2.19b).
- We were able to create the choice of viewing packet-switches and circuit-switches in different layers using an innovative procedure for discovering virtual-packet-links. In the next Chapter we will explore an application that treats the switches as different layers but still under common-control.
- In support of our larger goals to converge packet-and-circuit network operation in WANs, this prototype was sufficiently evolved to emulate WAN structure (unlike the previous two prototypes).
- Finally we were able to demonstrate control over a wide variety of data-plane flow-identifiers (L1-L4) and mappings that showed the power of this approach as a multi-layer control plane.

Prototype #3 implements most of the features of our architectural approach – the common-flow and common-map abstractions via a unified-control plane. It therefore gives us an opportunity to compare and contrast our work, to industry solutions for common-operation of packet and circuit switched networks. In the next chapter, we validate the simplicity and extensibility claims we made for our unified-control-architecture in Ch. 1 by comparing our work these industry solutions.

2.4 Summary

In this chapter, we gave details of the design of the two abstractions we introduced in Ch.1, as part of our unified-control architecture for packet and circuit network control.

We first discussed the common-flow abstraction. We showed what the flow-abstraction means in the context of packet-switching, and then showed how it can apply equally well to circuit-switching and different kinds of circuit switches. We then gave design details of a common switch-API to manipulate the data-abstraction of flow-tables in both packet and circuit switches. Importantly, we showed how the data-plane construct of a virtual-port can flexibly map packet and circuit flows to each other.

Next, we gave design-details of the common-map abstraction. We briefly discussed the representation of the common-map and how it can be built and maintained. We detailed the discovery of packet and circuit links which is an important part of the puzzle when building a common-map. And we outlined procedures via which the common-map can be represented as single or dual layers/topologies. Finally we discussed a set of network-API calls that are needed to manipulate the common-map.

We then delved into instantiations of the common-flow and common-map abstractions in the OpenFlow protocol and a network-operating-system called NOX. We presented three prototypes that helped us progressively validate our design constructs. Two early prototypes focused on the common-flow abstraction for packet-switches working with different kinds of circuit-switches (wavelength and time-slot based). The third, more complete prototype focused on the common-map abstraction and building a testbed that emulates WAN structure. We will use the latter in Ch. 3 to validate the simplicity and extensibility claims of our control architecture.

Chapter 3

Simplicity & Extensibility of Architecture

We proposed a unified control architecture for packet and circuit networks based on two abstractions: the common-flow and the common-map abstraction. We discussed an implementation of the architectural approach using OpenFlow and NOX, and presented several prototypes that helped us understand and refine our implementation. In this chapter we wish to validate the simplicity and extensibility claims of our architectural approach. Unified control over packets and circuits enables us to develop new networking capabilities that benefit from the strengths of the two switching technologies.

Thus, the first step of our validation methodology involves developing a new network capability; in this work, we choose to develop the ability for the network to treat different kinds of traffic differently, using both packets and circuits at the same time. Specifically the network treats voice, video and web traffic differently in terms of the bandwidth, latency and jitter experienced by the traffic, as well as the priority with which traffic is re-routed around a failure. We implement this capability on an emulated WAN using the full pac.c prototype discussed in Sec. 2.3.3. The second step of the validation process involves comparing our implementation, to an implementation which would use the current state-of-the-art industry-standard control-plane solution.

In support of our simplicity claim, we find that our implementation requires *two orders* of magnitude less code than the industry-standard solution. Lines-of-code differing by orders of magnitude are good indicators of the complexity in developing and maintaining one code-base compared to another. And so we conclude that our control-architecture enables simpler implementations of control-applications.

In support of our extensibility claim, we first discuss the two reasons that make our solution extensible: a) full visibility across packets and circuits; and b) the ability to write control applications in a centralized way, where the state-distribution mechanisms have been abstracted away. We find that the current industry standard supports neither of the two; which is why it is not surprising that even with two orders of magnitude more code, the industry solution *cannot replicate* the network capability we demonstrate. Next, as further proof of extensibility, we discuss three other examples of applications enabled at the intersection of packet and circuit switching: dynamic packet-links; variable-bandwidth packet links; and unified-routing.

Finally we discuss three deployment challenges faced by *any* unified control solution for packet and circuit networks. The challenges include: a) the reluctance of network operators of packet and circuit networks to share information with each other; b) the conservative nature of transport network operators towards automated control planes; and c) the conservative nature of IP network operators towards increasing load on distributed routing protocols. As final validation of the simplicity and extensibility of our work, we propose solutions based on our control architecture to *all* the deployment challenges mentioned above.

3.1 Demonstration of New Network Capability

The goal of the network function (or capability) we have chosen to implement is ‘to treat different kinds of traffic differently’. We first describe what kinds of traffic we deal with, and briefly why we wish to give them different treatment. We identify how we achieve our goals by leveraging the benefits of both packets and circuits, as well as our common-global view of both networks and common control over both switching technologies (Fig. 3.1). We then briefly describe the software architecture of our system and network application, and give details of our demonstration [52, 53].

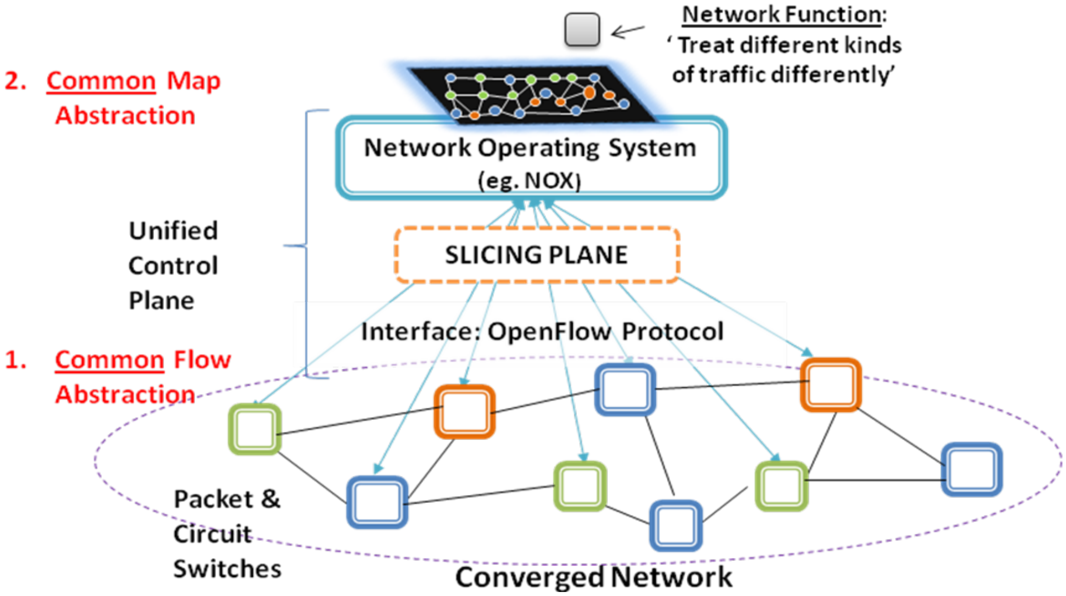

Network Function:
' Treat different kinds of traffic differently'
2. Common Map Abstraction
Network Operating System (eg. NOX)
Unified Control Plane
SLICING PLANE
Interface: OpenFlow Protocol
1. Common Flow Abstraction
Packet & Circuit Switches
Converged Network

Figure 3.1: Example Network Function

3.1.1 Example Network Function

The network function is designed to provide differential treatment to three types of traffic – voice, video and web. Table 3.1 shows the differential treatment we designate for these types of traffic with respect to delay, jitter and bandwidth experienced by the traffic, and priorities given to re-routing said traffic around failures.

It is important to note that this is just an example of a function that we (as a network operator) wish for our network to perform. It is by no means the only application that can be performed, nor is it the only kind of treatment that can be delivered to these types of traffic. We have simply chosen this treatment to demonstrate a reasonably involved network function. Other network operators may wish to provide different treatment and that too can be implemented easily with our architectural approach.

Differential Treatment: For Voice-over-IP (VoIP) traffic, it is important to minimize latency i.e. the delay experienced by packets carrying voice data. This is because voice communication (eg. a Skype or Google-Voice call) is usually interactive and bi-directional, which makes it important that both communicating parties receive packets as quickly as possible in order to respond. At the same time, bandwidth required by VoIP traffic is usually low.

Traffic-type	Delay/Jitter	Bandwidth	Recovery
VoIP	Lowest Delay	Low	Medium
Video	Zero Jitter	High	Highest
Web	Best-effort	Medium	Lowest

Table 3.1: Example Network Control Function

In contrast, streaming video involves mostly one-way communication - for example, Netflix servers streaming video to an end-user; or the streaming of a live-event (like sports) by a channel (like ESPN3). In such cases, overall delay matters, but less than in bidirectional voice communications. What matters more is the variation in delay (or jitter) as such variations can cause unwanted pauses or artifacts when rendering the video on the client's device.

The bandwidth experienced by video traffic is important as well. Note that when we talk about bandwidth, we are *not* referring to the data-rate of the video stream[†]. Instead we refer to the share of *link* data-rate 'seen' by the video traffic as it propagates through the network. If this share is constant, then as more video streams are added to the links, each stream experiences less data-rate, congestion develops and video-packets are dropped. While voice traffic can in most cases tolerate some packet loss (eg. users can repeat what they said), such packet-losses can be devastating to user experience when

[†] This data-rate typically varies depending on the amount of visual change in the video and the compression mechanism (eg. I-frames in MPEG video have higher instantaneous data-rates than B or P frames).

watching video (pauses, artifacts such as black-spots or discoloration). And so, not only do we give a larger share of bandwidth to video traffic, we ensure that as more video is added to the network, the bandwidth experienced by the traffic *remains* high in order to avoid or minimize video-packet losses.

Finally we deem web-traffic[†] to be best effort, in that we do not make any special requirements for delay or jitter. We choose to allocate medium levels of data-rate for this traffic class (more or similar to voice but less than video). Importantly, we do not ensure that data-rate 'seen' by web or voice traffic remains at the same levels at all times. And when failure of some kind happens in the network (link or node), we prioritize the recovery of all video traffic, over voice or web traffic.

Approach: Our solution for realizing the aforementioned network capability has four distinct parts (Fig. 3.2); each of the parts is made possible by our SDN-based unified control architecture for packet and dynamic-circuit switching:

1. <u>Dynamic Service-Aware Aggregation</u> - Aggregation is necessary in WANs so that core-routers have more manageable number of rules in their forwarding tables (a few hundred thousand instead of millions). Aggregation can take the form of IP supernetting (or CIDR; eg. using /20s instead of individual /24s) or by the insertion of labels and tags. Often, aggregation is manually configured, static and error-prone. But our switch-API (OpenFlow) allows great flexibility in defining flow granularity: We can dynamically and programmatically perform aggregation in packet-switches simply by changing the definition of a flow [54]. For example, all the flows from one customer may take the same route in a network; then we can perform aggregation simply by entering a single flow-table entry to match on the customer's *source*-IP address (instead of matching different destination-IP addresses). Note that the packets themselves do not change, just their representation in the packet-switches' flow table changes. And so, in the core we can collectively reference all flows with a single aggregated flow bundle (using the source addr). We could also perform supernetting or label insertion if desired. We can go further by differentiating between traffic types

[†] In this work we still classify voice and video traffic over HTTP (eg. You-Tube or Netflix) as web traffic.

from the same customer (as required by our control function), by creating separate bundles (or aggregates) that match on the customer's source IP address *and* the tcp/udp port in the packet header – eg. web (tcp port 80), voice (tcp port 5060) and video (udp port 1234) traffic.

2. Mapping & Bandwidth: Next we *map* the aggregated packet-flows to different circuit-flows; in order to give different service quality to each bundle in the transport network. We assign *different bandwidths* for the circuit-flows, thereby partially satisfying the bandwidth requirements in Table 3.1. For example, from a single 10Gbps packet interface, we map the three traffic bundles to three circuits, with the video circuit assigned 5 Gbps, the voice traffic 1 Gbps, and the web traffic-circuit assigned the rest. Additionally we selectively *monitor the bandwidth usage* of the circuit-flows, and as usage varies, we *dynamically resize* the circuits according to our control-function needs. For example, as more video traffic passes through the link, the control function requires that the video traffic collectively still experiences high bandwidth. And so, by selectively monitoring the bandwidth consumption of the video-circuit, we can dynamically change its size when sustained video-traffic surge is observed. Such additional bandwidth can, for example, be borrowed from the web traffic-circuit to ensure low video-packet loss.

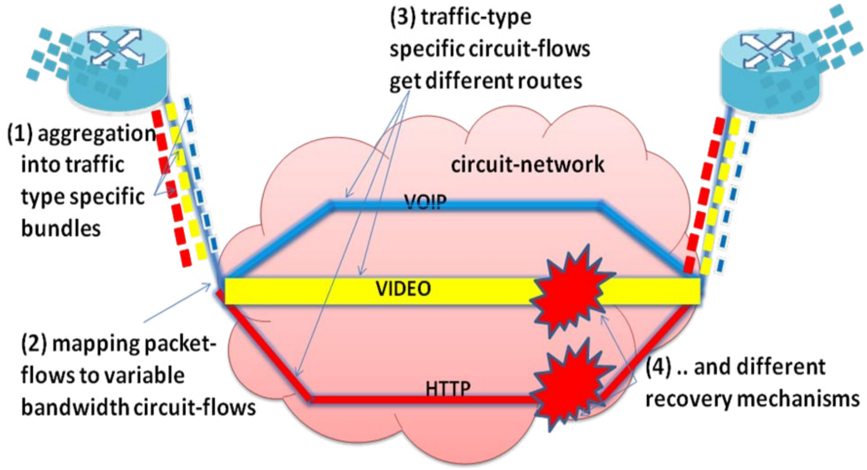

(3) traffic-type specific circuit-flows get different routes
(1) aggregation into traffic type specific bundles
circuit-network
VOIP
VIDEO
HTTP
(2) mapping packet-flows to variable bandwidth circuit-flows
(4) .. and different recovery mechanisms

Figure 3.2: Control Function Implementation Using Packets & Dynamic Circuits

3. Routing for Delay & Jitter: We can tailor a circuit to have *routing characteristics* beneficial for an application or service. For instance, the VoIP traffic-circuit can be routed over the shortest path in the physical fiber topology. In contrast, the video-bundle can be routed over a path with zero-jitter by bypassing intermediate packet-switches (that would introduce jitter) between source and destination routers.
4. Prioritized Recovery: Finally, once we have global knowledge and centralized decision making, we can recover from network failures so as to meet application needs. For example, video traffic could be circuit-protected with pre-provisioned bandwidth (ensuring fastest recovery), while all voice could be dynamically re-routed by the controller before any web-traffic is re-routed.

Software Architecture & Implementation: Next we present the software architecture of our implementation on top of the common-map abstraction. We highlight a few features of the software architecture with respect to discussions from previous chapters:

- First, note that Fig. 3.3 is an extension of Fig. 2.19, where we have added network-applications on *top* of the changes we previously made to NOX. To recap, such changes were required to create the common-map abstraction; and they included changes to the OpenFlow protocol; modules for *Circuit-Link-Verification* and circuit *Topology*; a *Circuit-Switch-API* for manipulating switches in the common-map directly; as well as a *network-API* for network-wide control functions.
- Second, we had previously stated that the common-map abstraction does not preclude any layering choice for packets and circuits. In this particular example we choose to treat the circuit and packet topologies in different layers (corresponding to the left-side of Fig. 2.12). As an example of a different layering choice, later in this chapter (Sec. 3.4) we discuss an example application that treats the topologies as one (in a de-layered way).

- Third, while flow databases (for both packet and circuit flows- see Fig. 1.11) can be a part of the common-map as they reflect network state, often the decision to retain such state in the controller is left up to the application-writer. Applications may decide *not* to keep any flow-state in the controller as such state can be voluminous and ephemeral (and therefore quickly get out of hand). In such cases, when there is change in network-state which requires changes to flow-state, the latter can be re-computed. But more slowly varying flow-state such as *aggregated* packet-flows and more manageable flow-state such as circuit flows (fewer cflows than pflows) may well be retained in the controller. Accordingly we choose to retain state of such flows in the *AggregationDB* and *cFlowDB.*

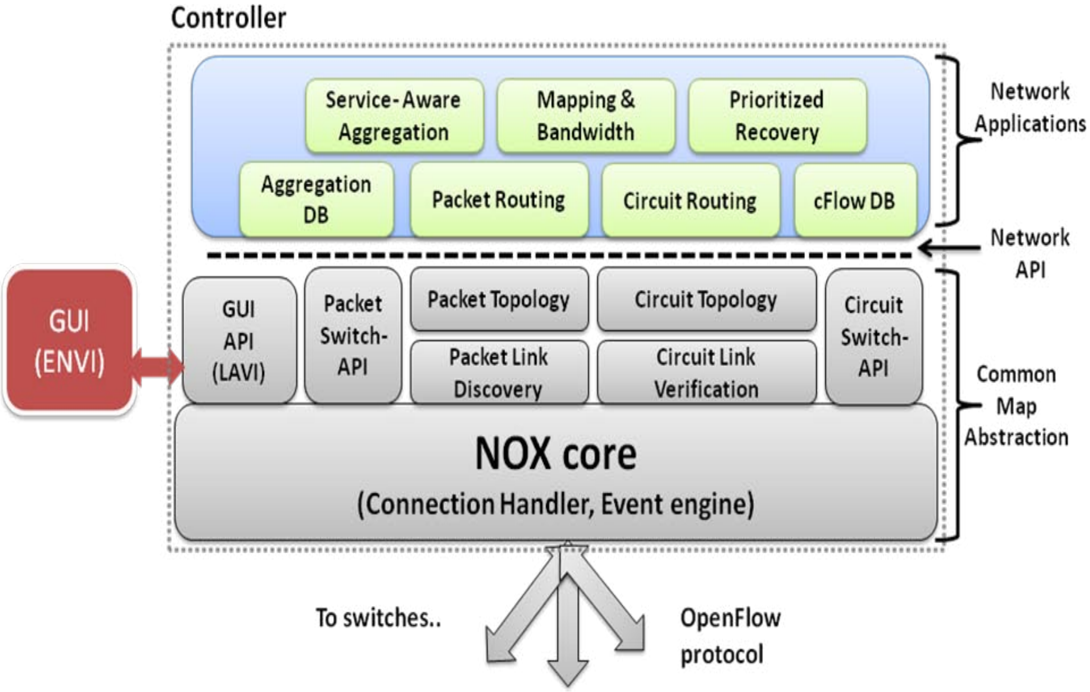


Figure 3.3: Software Architecture

Since we chose to retain packet and circuit layering in this example, we have separate routing modules for packets and circuits. The *Packet-Routing* module is responsible for finding shortest paths over the packet topology for aggregated bundles of packet-flows.

Such aggregated bundles are created by the *Service-Aware Aggregation* module, which classifies incoming packets into traffic-type specific bundles and informs the *Mapping & Bandwidth* module of desired circuit-flow characteristics. The latter application maps aggregated packet-flows to circuit-flows, and determines their bandwidth allocations. It is also responsible for constraining the *Circuit-Routing* module to determine routes for the circuits according to traffic-specific requirements (delay, jitter, bandwidth etc.). Additionally it monitors certain circuit flows for their bandwidth usage and resizes the circuits according to application needs. Finally the *Prioritized Recovery* module re-routes circuit-flows upon link-failure according to the network-function guidelines.

3.1.2 Experimentation

The network application we discussed in the previous section is demonstrated on our full pac.c prototype. Details of this prototype and the way in which it emulates WAN structure (Fig. 3.4a) were discussed in Sec. 2.3.3. We also created two GUIs that display network state for the emulated WAN in real-time. Importantly, the GUIs present *different views of the same network* – the packet layer in the upper GUI in Fig. 3.4b (including physical and virtual links); and circuit layer in the lower GUI (with only physical links)[†].

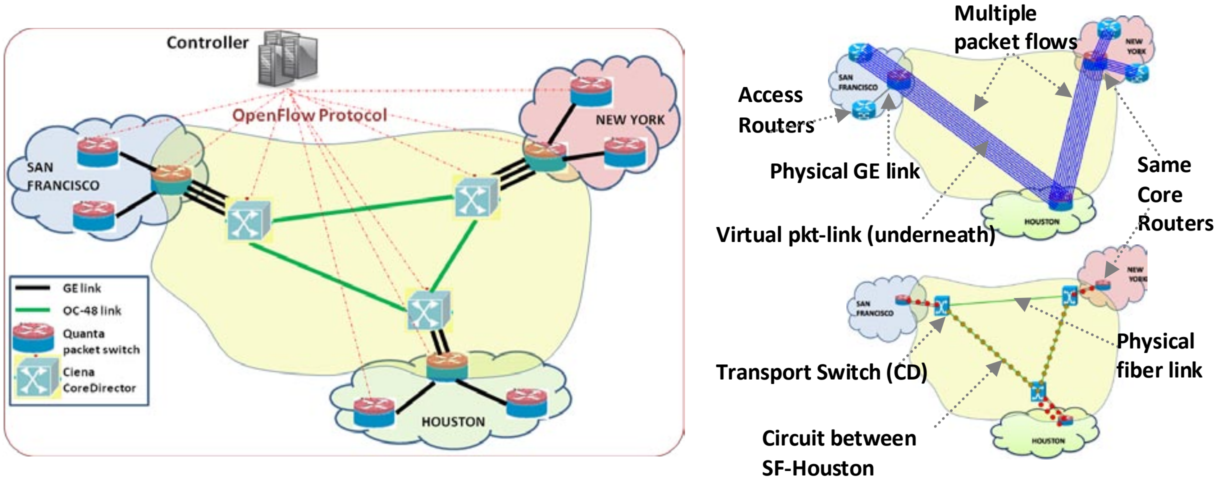


Figure 3.4: (a) Emulated WAN structure (b) GUIs displaying real-time network state

GUI-views: The upper GUI in Fig. 3.4b shows packet-switches in three city-PoPs. Physical GE links connect Access-Routers to Core-Routers within each PoP. Between cities, the links shown in the upper GUI correspond to *virtual* packet-links. These virtual links are supported by circuits-flows in the underlying circuit-layer (shown in the lower GUI). The packet-layer view in the upper GUI also shows packet flows currently routed in the network – for example, multiple packet-flows are shown in Fig. 4.3b routed from an SFO Access Router to a NY Access Router via the Core Routers in SFO, Houston and NY(obscuring the view of the links underneath). Note that the upper GUI does *not* show any circuit-switching equipment.

The lower GUI in Fig. 3.4b displays the transport-switches – i.e. the CoreDirectors (CDs); the physical fiber topology; and *only* those packet switches that are physically connected to the CDs. As a result, note that the Core-Routers in all three cities appear in *both* GUIs – in the upper one they are at the ends of the virtual-packet links over the wide area; and in the lower one, they are physically connected to the CDs. The lower GUI also displays circuit-flows – for example two circuits are shown, one each between SF-Houston and Houston-NY, which support the virtual-packet-links and packet-flows shown in the upper GUI. To further clarify, note that a virtual-packet link between core-routers (in the upper GUI) can be supported by single or multi-hop circuits in the transport network (lower GUI view), as shown in Fig. 3.5.

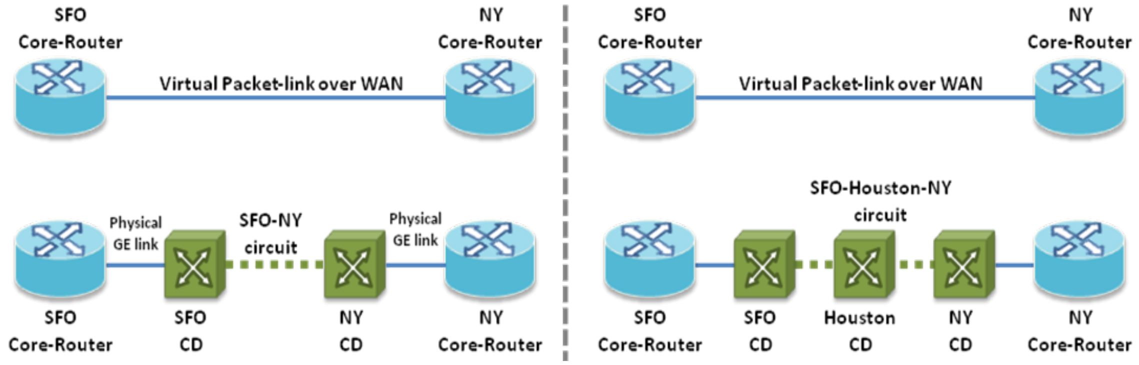


Figure 3.5: GUI views

Application-Aware Aggregation: Initially, traffic consists of 10 flows from customer 1. The flows are routed from SF to NY via the Houston Core-Router. Without aggregation, each flow is treated individually by the network, based on the shortest path between source and destination. In other words, there are 10 flow-table entries, for each flow's destination IP address, in each packet-switch along the path. The *Service-Aware Aggregation* module dynamically performs aggregation by creating three bundles (aggregated packet-flows), one for each traffic type from the same customer, by matching on the customer's source IP address *and* the tcp/udp ports in the packet header (packet GUI in Fig. 3.6a).

Note from the circuit GUI in Fig. 3.6a, these 3 application-specific bundles still take the same route in the core, i.e they go from the SF Core Router to the SF CD, then over the circuit to the Houston CD; then to the Houston Core Router; where it gets switched back to the Houston CD to go over the circuit to the NY CD and finally to the NY Core Router. In other words, all 3 bundles are still treated the same way in the network.

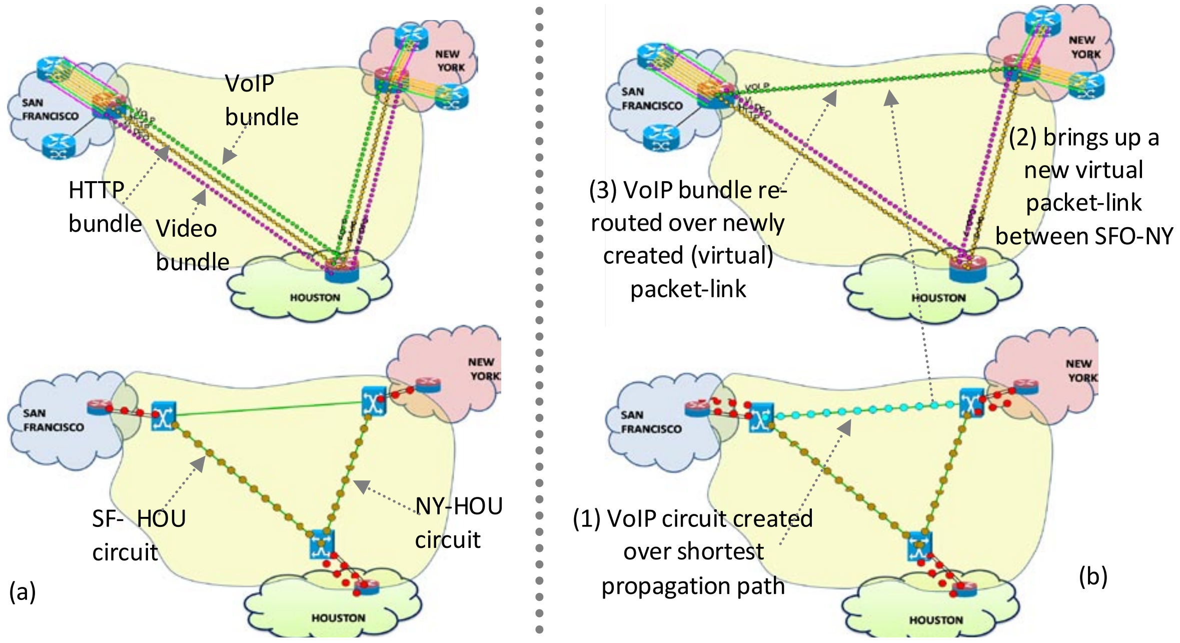

SAN FRANCISCO
NEW YORK
HOUSTON
VoIP bundle
HTTP bundle
Video bundle
SF- HOU circuit
NY-HOU circuit
(a)
(3) VoIP bundle re-routed over newly created (virtual) packet-link
(2) brings up a new virtual packet-link between SFO-NY
(1) VoIP circuit created over shortest propagation path
(b)

Figure 3.6: (a) Service-aware Aggregation (b) VoIP treatment

Re-routing VoIP Traffic: The *Service-Aware Aggregation* module informs the *Mapping & Bandwidth* module of the low-latency and bandwidth needs for the VoIP bundle. The latter uses the *Circuit Routing* module to pick the shortest propagation path in the circuit network that satisfies the bandwidth requirements (which in this case is the direct fiber linking the transport switch in SF to the one in NY). It then dynamically creates a circuit between SF and NY that in-turn brings up a virtual packet-link between the Core Routers in SF and NY (Fig. 3.6b). It does so by transparently transporting link-discovery packets sent-out by the controller from the SF Core-Router to the NY Core-Router[†]. Note that in Fig. 3.6a no such link existed between SF and NY Core-Routers in the packet-layer. The *Service-Aware Aggregation* module then selectively re-routes just the VoIP bundle over the newly-created packet link (Fig. 3.6b).

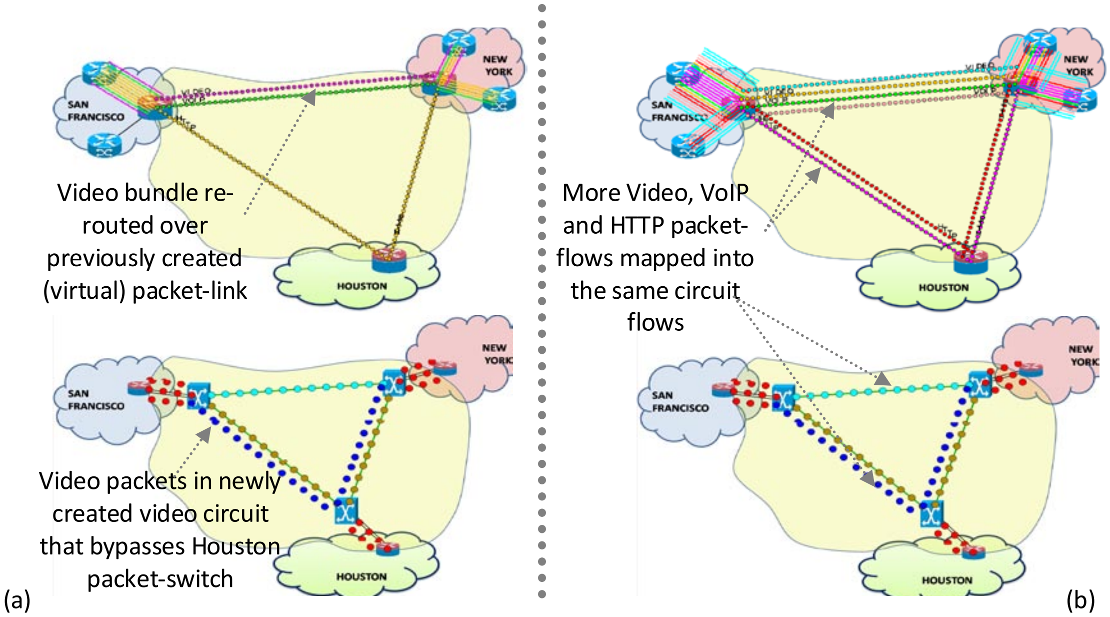

SAN FRANCISCO
NEW YORK
HOUSTON
Video bundle re-routed over previously created (virtual) packet-link
Video packets in newly created video circuit that bypasses Houston packet-switch
More Video, VoIP and HTTP packet-flows mapped into the same circuit flows
(a)
(b)

Figure 3.7: (a) Video treatment (b) More aggregates mapped

Re-routing Video Traffic: For video, the *Circuit Routing* module picks a non-shortest path circuit (delay less important) with a higher bandwidth requirement (than VoIP). Such a circuit is dynamically created between SF and NY, after which the *Aggregation* module reroutes the video-bundle. Importantly this video-bundle goes

[†] See discussions in Sec. 2.2.2 and 2.3.3

through the Houston PoP but stays in the circuit layer, bypassing the Houston Core Router, and therefore avoids potential jitter in that router.

Note that in the packet-GUI in Fig. 3.7a, the video-bundle appears to go over the *same* virtual-packet-link between SF and NY that was created for the VoIP bundle. In reality the VoIP and Video bundles use the same physical-port on the SF Core Router but end up taking different physical paths in the circuit network (as discussed in Sec. 2.3.3); ones that are suited for the traffic-type.

Also note that the web-traffic still uses the original SF-Houston-NY path in the packet-layer and is therefore treated as best-effort; i.e. no special circuits are created for web traffic. As more traffic arrives for all three kinds of traffic, from the same or different customers, they are similarly aggregated and mapped into the circuit-flows created for them (see upper GUI in Fig. 3.7b).

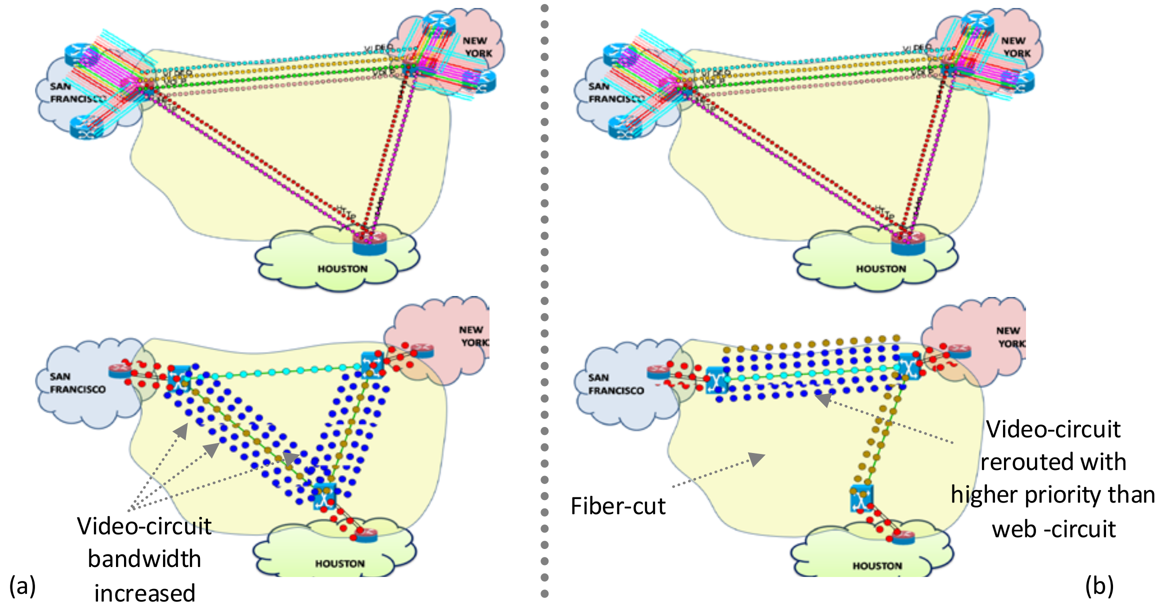

SAN FRANCISCO
NEW YORK
HOUSTON
Video-circuit bandwidth increased
(a)
SAN FRANCISCO
NEW YORK
HOUSTON
Fiber-cut
Video-circuit rerouted with higher priority than web -circuit
(b)

Figure 3.8: (a) Bandwidth increased for Video (b) Prioritized Recovery

Monitoring & Recovery: Also as traffic increases, the *Mapping & Bandwidth* module monitors the bandwidth usage of the circuits to accommodate for traffic growth. In Fig. 3.8a, the video-circuit is dynamically re-sized when sustained traffic surge is observed, as per the requirement of having video-traffic experience high-bandwidth. Finally, the *Prioritized Recovery* module dynamically re-routes circuits upon failure. In Fig. 3.8b, when the fiber-link between SF and Houston breaks, all video-traffic based circuits are re-routed *before* re-routing the circuits meant for best-effort web traffic.

Conclusions: In this section, we achieved the first step of the validation methodology outlined in the introduction. We discussed a new network capability and showed how it could be implemented using packets and dynamic-circuits commonly controlled within our unified control architecture. We showed how voice, video and web traffic can be classified, aggregated and mapped into specially routed, dynamically created, monitored circuits to achieve differential treatment. In the next two sections, we move to the next step of our validation methodology, by comparing our work with ways in which such capabilities could be replicated using current industry solutions.

3.2 Comparison to Existing Solutions: Simplicity

In the previous chapters we claimed the benefits of simplicity and extensibility of our architecture compared to existing industry solutions. In this section we validate our simplicity claim. But first we explain what we are comparing our solution to, by detailing the current industry solution. We investigate the requirements for providing a service or network capability such as one described in Sec. 3.1, across packets and circuits with such industry-based control architecture.

To compare between our implementation and one based on industry solution, we contrast the Lines of Code required to implement the function. Why do we use Lines of Code? Because orders of magnitude difference between the lines-of-code for different software projects can estimate the complexity of the project, in terms of the amount of effort required to develop the code-base as well the effort required to maintain or change the code-base after release [59]. We count the physical lines-of-code (without blank and comment lines) using CLOC [60]. Tables from the analysis are shown in Appendix D.

3.2.1 Implementation with Industry Solution

Today's packet and circuit networks are not converged networks. Core packet networks (IP/MPLS) are planned, designed and operated separately from core circuit networks (Transport networks), even if both networks are owned by the same service provider.

IP Networks: Today's IP core-networks are no longer plain-vanilla IP networks – MPLS is widely used. In fact there isn't a single Tier-1 ISP today that does *not* run an IP/MPLS network [56]. We make the observation that MPLS† introduces the concept of 'flows' in IP networks [57]. In principle, packets can be classified into Forwarding Equivalence Classes (FECs); all packets in the same FEC are forwarded the same way via Label Switched Paths (LSP); and accounting/resource management can be performed on an LSP level; making FEC+LSPs similar to our 'flow' definition (Sec. 1.4.1).

Transport Networks: Today's transport networks are divided into vendor-islands (Fig. 3.9). The switches within a vendor-island support proprietary control and management interfaces that only work with the vendor's EMS/NMS. In other words, they do not interoperate on the control plane with switches in other vendor islands. Note that the multiple vendor islands shown in Fig. 3.9 still represent a *single* carrier's transport network, and the operation of such a network is highly manual (human-driven NMS).

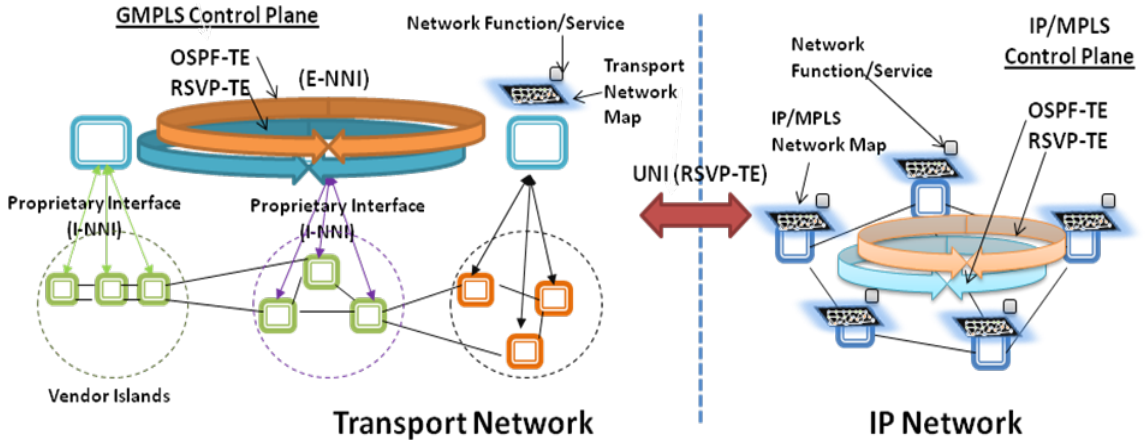


Figure 3.9: Industry Solution for IP and Transport Network Control

† We will have much more to say about MPLS, the flow-abstraction and the map-abstraction in Ch. 5

The industry has developed an interoperable automated, control plane solution (GMPLS) based on the MPLS control plane protocols. Such protocols were adopted and extended for circuit switching features by the standards bodies (IETF and ITU). GMPLS protocols were used to *stitch* together the vendor-islands as a layer above the proprietary interfaces, which remained the same as before. Collectively the proprietary interfaces are known as I-NNI and the stitching interface is the E-NNI. Finally, an additional interface known as the UNI was defined, for client-networks (such as the IP network) to interface with and request services from the transport network[†].

So collectively the MPLS/GMPLS/UNI/NNI solution would look like Fig. 3.9. As noted in Sec. 1.5, this industry solution (GMPLS and UNI) has never been commercially adopted [26, 27]. But it is the only solution proposed by the industry for packet-circuit network control, and so we use it to investigate and compare to our solution.

Software Stack: The IP/MPLS network is a fully-distributed network where each router makes decisions on packet forwarding based on a variety of network control-functions (Fig. 3.10). There are two consequences of such design:

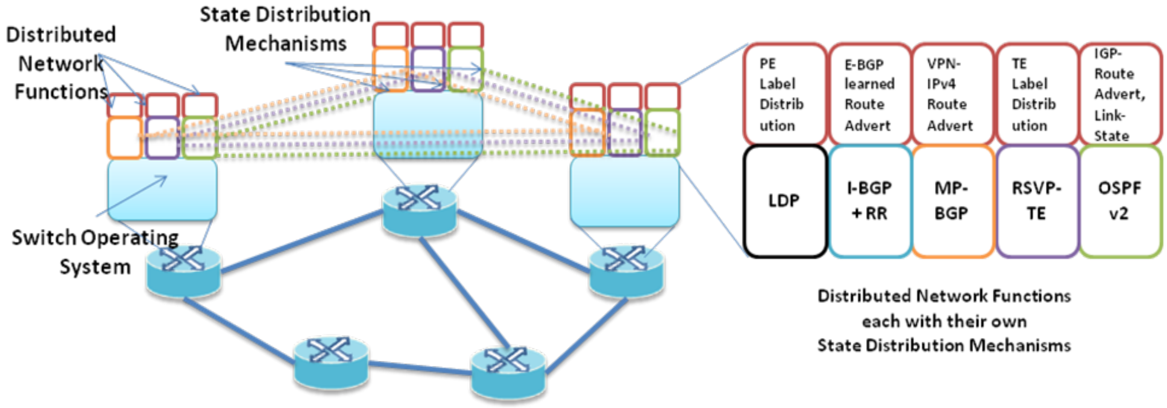

Distributed Network Functions
State Distribution Mechanisms
Switch Operating System
PE Label Distribution
E-BGP learned Route Advert
VPN-IPv4 Route Advert
TE Label Distribution
IGP-Route Advert, Link-State
LDP
I-BGP + RR
MP-BGP
RSVP-TE
OSPF v2
Distributed Network Functions each with their own State Distribution Mechanisms

Figure 3.10: Software Stack in Fully-Distributed Control Planes

1. Complexity: Since each router in the network largely makes its own decision, network-functions/ applications/services are then in most cases, *necessarily* implemented in fully distributed-ways, in each and every router in the network. Such distributed network functions need to implement their own distribution mechanisms,

[†] For more details on the origins and choices made for GMPLS, see the Related Works Section in Ch. 6.

and depend on them for proper working, leading to exposure to distributed state and subtle interactions with the mechanisms. This in turn makes it *harder to implement* the function (more code to touch), get it right (testing and verification) and maintain it over time (as function requirements change and/or new functions are introduced). As an example, from Fig. 3.10, the network functions of distributing IPv4 address, MPLS labels and link-state are handled by a variety of protocols each with their own characteristics and nuances. And these are just for unicast IPv4. There are many more protocols for handling state-distribution for other purposes like IPv6, multicast, etc.

2. Extensibility: In most cases, changes to a network function requires changes to the distribution mechanism[†] or the creation of an entirely new distribution mechanism[^]. This hurts *extensibility* of such control-architectures, which we will discuss in the next section. Here we merely wish to point out again why distributed features are tied to the distribution mechanisms that support them.

Lines-of-Code: Our control function (in Sec. 3.1) requires the network to a) differentiate between different types of traffic (voice, video, web) to create traffic-type specific aggregates and b) forward the aggregates (bundles) differently i.e. not necessarily along the shortest path in the network. Given the industry standard solution and structure shown in Fig. 3.9, we need the following protocols and feature implementations:

- IP-network: An IP/MPLS network would require Policy-Based-Routing [61] for differentiating between traffic types and forwarding them to different traffic-type specific tunnels, created using Diffserv-aware MPLS Traffic Engineering (DS-TE) mechanisms [62]. The DS-TE tunnels can then be routed along paths in the network that are not necessarily the shortest ones. PBR and DS-TE are typically implemented by router vendors. We were unable to find an open-source implementation of either. While PBR is a local-hack in a router at the source-end of an MPLS-TE tunnel, DS-TE is a distributed application dependant on the distribution mechanisms of OSPF-TE

[†] The original RSVP protocol (hosts signaling for services - not used anymore), has undergone no less than three such 'extensions' – first it was given TE extensions for MPLS, then extensions for transport switches in GMPLS, and then it was extended again for supporting the UNI/E-NNI interfaces.

[^] When the MPLS control plane was extended for GMPLS, a new protocol, LMP (RFC 4204) was created in addition to extensions for OSPF and RSVP.

and RSVP-TE. Thus we present *lower-end estimates* for the number of lines of code that need to be touched for implementing the packet-part of our control-function. We have counted the lines-of-code (in Appendix D) for two open-source projects that have implemented OSPF-TE and RSVP-TE. The Quagga project [63] implements the former in 34,244 lines-of-code, where we have only accounted for the 'ospfd' implementation and not any of the other routing protocols that are part of Quagga. Also since the Quagga suite does not include RSVP-TE, we use the IST-Tequila project implementation for RSVP-TE [64, 65] (49,983 lines of code).

- UNI Interface: To implement our control function using packets and circuits (Fig. 3.9), one or more routers would have to request services from the transport network using the UNI interface. The UNI is typically instantiated using a signaling protocol like RSVP-TE with suitable extensions [22]. We count 9,866 lines-of-code from an implementation of the OIF UNI protocol from the IST-MUPBED project [66, 67].
- Transport-Network: Finally the GMPLS protocols for the transport network control plane include further extensions of the MPLS control plane protocols – OSPF-TE and RSVP-TE. Such extensions have been standardized [20]. We use the DRAGON project [68, 69] for representative open-source implementations of the GMPLS protocols – 38,036 lines-of-code for OSPF-TE and 43,676 for RSVP-TE. Note that we are ignoring the code required for the proprietary interfaces in the transport network vendor islands (the I-NNI protocols).

If we put together all the lines of code across packet and circuit control-planes, we get a total of 175,805 lines of code for *just* the state-distribution mechanisms. In addition, more lines-of-code would be required to implement the distributed-control logic that performs the task required by the control function. These logic components would include PBR, DS-TE, a CSPF algorithm for routing of primary/backup circuit-paths, and glue-code for translating GMPLS messages to the proprietary interfaces. Based on our experience with implementing distributed protocols any one of the mentioned logic components would at a minimum require 5-10k lines-of-code, pushing the total lines-of-code to well over 200,000. Approximate totals for the lines-of-code are depicted in Fig. 3.11 below.

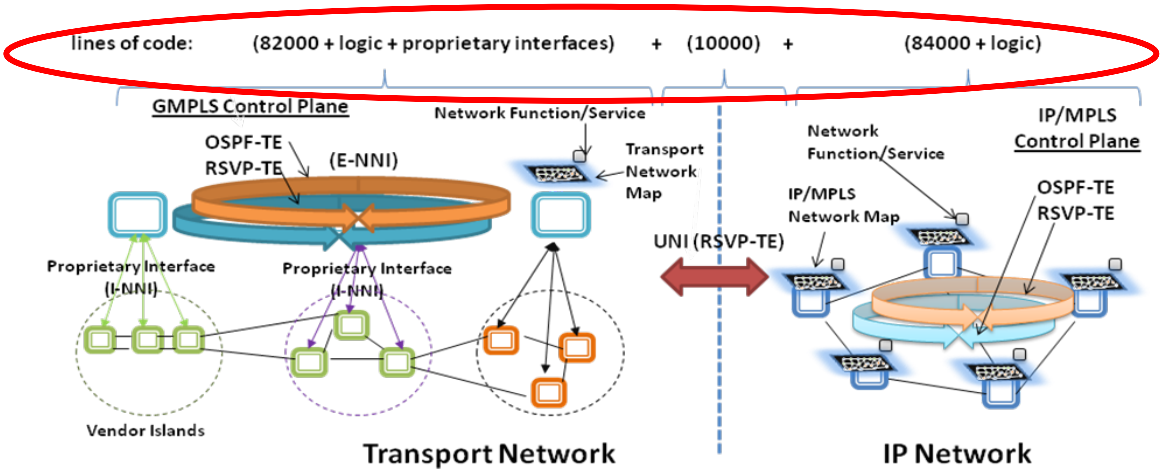


Figure 3.11: Lines-of-Code in Industry Standard Solution

It is worth mentioning that the 175,000 + ‘x’ lines of code only relate to the network applications, which include the function-logic as well as the distribution mechanisms. These applications are implemented on top of some base code – for example, the Quagga software suite is based on the zebra-daemon and related libraries. This gives us about 52,000 lines of code for the Quagga-base code, built on top of the Linux kernel. Finally, Fig. 3.12 also shows a closed-source implementation of router software from vendors such as Cisco (IOS) and Juniper (JUNOS). Such implementations of functions and services together with distributed protocols and the base operating system for the router have been known to total approximately 20 million lines-of code (in each router) [70].

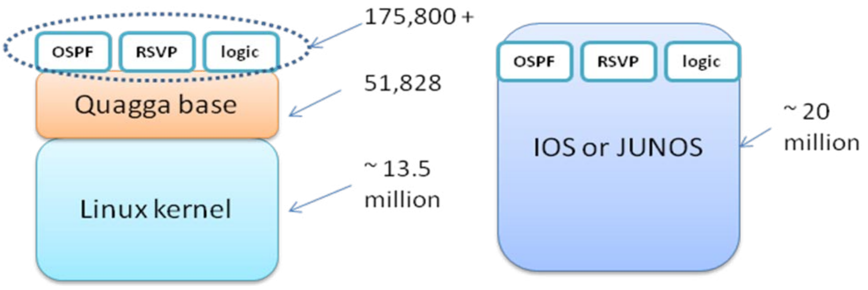


Figure 3.12: Source Lines-of-Codes in Distributed Approaches

3.2.2 Implementation with Unified Control Architecture

Simplicity comes from not having to implement each control functions as a distributed system, and not having to worry about interactions with state-distribution mechanisms. There are two kinds of state-distribution mechanisms in SDNs (Fig. 3.13a):

1. One is between the controller and the switches. In SDNs, the OpenFlow protocol could be used for this purpose where the controller communicates directly with the switches one-on-one. Since decision making is no longer the switch's responsibility, the need for distributed routing and signaling protocols is removed within a controllers' domain. And so such protocols are eliminated.
2. The second distribution mechanism is between the multiple physical servers that make up a controller. In our work we use NOX as the network OS. NOX is meant to run on a single server. Commercial network OSes running on multiple servers have additional functionality for distributing state between these servers thereby increasing the lines of code for the Net-OS. Importantly, these additions only need to be done once and are then abstracted away from multiple (centralized) applications that can be written on top of the common-map abstraction. With proper implementation, these state-distribution mechanisms would not count as part of feature development cost as they do not need to be changed or touched in order to implement a new feature.

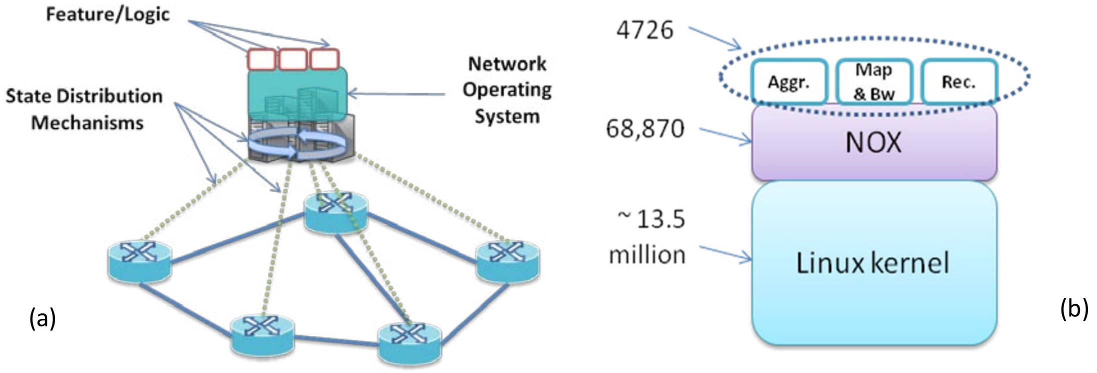


Figure 3.13: (a) Software Stack and (b) Source Lines-of-Code in SDN

The advantage of our approach can be easily seen in Fig. 3.13b. Our control applications [71] are implemented on top of NOX. NOX itself is based on the Linux kernel (millions of lines of code [59]). The base code of NOX is roughly 67,700 lines of code (Table 3.6 and [51]) to which we have added 1100 lines to support circuit-switching. While the base line-of-code for NOX is on the same order of magnitude as the Quagga-base code (Fig. 3.12), it is completely different in functionality and purpose from Quagga. Importantly, the SDN based approach of our control architecture (Fig.3.13a), helps us implement the control function in only 4726 lines-of-code – *two orders of magnitude less* than the industry solution (details in Appendix D). This significant difference validates the simplicity claim of our control architecture. Our implementation is not production-ready. But being two orders of magnitude less than distributed implementations leaves plenty of room to grow.

To summarize, the objective of this exercise was to show the simplicity of writing *applications* in a non-distributed way afforded by the common-map abstraction. The objective was not to count *all* the lines-of-code in a system; but to emphasize the code-lines that potentially need to be dealt with when a new function is introduced in the network.

3.3 Comparison to Existing Solutions: Extensibility

In the previous section we found that implementing control-applications with our solution can be two orders of magnitude *simpler*, when measured in terms of lines-of-code, compared to implementing with current industry solutions. In this section we show how our solution affords greater *extensibility* compared to the same industry-solutions.

We show extensibility by first discussing why the common-map abstraction is the right abstraction for writing applications across packets and circuits, as opposed to the UNI interface. And then we discuss other applications enabled at the packet-circuit interface and show why they are hard or impossible to implement with the UNI interface.

3.3.1 Why the Common-Map is the Right Abstraction!

It is readily obvious from Fig. 3.14 that the lines of code involved in creating an application across packets and circuits using current industry solutions are high – nearly 200k. But what is less obvious, is that *even with* 200k lines-of-code, the network-function and demonstration presented in Sec. 3.1 *cannot be exactly reproduced*. Here is why:

Lack of Visibility: The primary reason is the lack of visibility across the UNI interface. The abstraction instantiated by the UNI interface is that of a black box. No information about either of the networks or their resources is exchanged across the interface in either direction. In other words, the IP network has no visibility into transport resources, and likewise the transport network has no visibility into IP services or traffic.

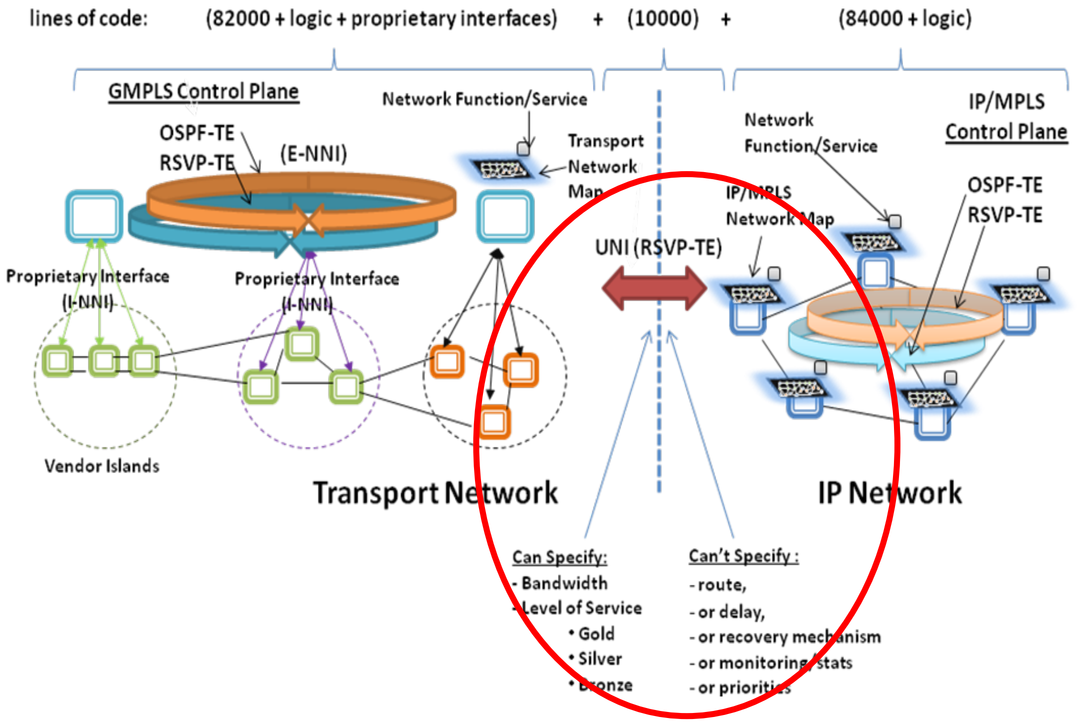


Figure 3.14: Issues with Industry Solution for Packet-Circuit Control

Table 3.2 shows the parameters used when a client (such as an IP Router) requests a connection (circuit) from the transport network using the UNI interface [72]. The shaded parts of Table 3.2 offer some insight into the kind of requests that can be made. An application in the packet-world can specify: traffic-parameters (which gets resolved into required-bandwidth for a circuit provisioned in the transport network); a level-of-service; and desired diversity of new circuit paths from existing paths originating from the same UNI. The rest of the parameters in Table 3.2 identify the end-points of the communication and other names, ids, labels and contracts associated with the connection.

What this basically means is that an application in the IP network *cannot specify any* of the following: circuit-route, delay, recovery mechanism, recovery priority, pre-emption, which circuits to monitor, or receive statistics from. But given full visibility, a*ll of these* were possible and demonstrated with our control-architecture in Sec. 3.1.

Attributes	Applicability	Call/Conn	Reference
Source TNA Name (M)	Cases 1 and 2	Call	Section 10.13.1.1
Source Logical Port Identifier (M)	Case 1	Connection	Section 10.13.1.2
Source Generalized Label (O)	Case 1	Connection	Section 10.13.1.3
Destination TNA Name (M)	Cases 1 and 2	Call	Section 10.13.1.1
Destination Logical Port Identifier (O-1, M-2)	Cases 1 and 2	Connection	Section 10.13.1.2
Destination Generalized Label (O-1, M-2)	Cases 1 and 2	Connection	Section 10.13.1.2
Local connection ID (M)	Cases 1 and 2	Connection	Section 10.13.1.4
Contract ID (O)	Case 1	Call	Section 10.13.4.1
Encoding Type (M)	Cases 1 and 2	Call/Connection	Section 10.13.2.1
Switching Type (M)	Cases 1 and 2	Call/Connection	Section 10.13.2.2
SONET/SDH, OTN or Ethernet traffic parameters (M)	Cases 1 and 2	Call/Connection	Section 10.13.2.3
Directionality (O)	Cases 1 and 2	Connection	Section 10.13.2.4
Generalized Payload Identifier (O)	Cases 1 and 2	Call	Section 10.13.2.5
Service Level (O-2)	Cases 1 and 2	Call	Section 10.13.2.6
Diversity (M)	Cases 1 and 2	Call/Connection	Section 10.13.3.1
Call Name (O-1,M-2)[1]	Cases 1 and 2	Call	Section 10.13.1.5
Bandwidth Modification Support(M)	Cases 1 and 2	Connection	Section 10.13.2.7

Table 3.2: UNI 2.0 Connection Setup Request Parameters (Table 5-11 from [72])

It is possible to define 'some' of the parameters mentioned above in an indirect way through the Service Level. Such classes of service are designated Gold, Silver, or Bronze and are *pre-determined offline* by the transport service provider. In other words, the packet–network (or any other client) is *limited* to the exact service-level definitions the transport network deems fit. This lack of visibility across the UNI interface coupled with the pre-supposition of services absolutely kills programmability.

Further, the translation of these levels of service into actual circuit-characteristics (route, delay, recovery etc.) is baked into the infrastructure via configuration of the UNI interface on the transport-side; and their distribution to other elements in the transport network by the dissemination mechanisms (GMPLS protocols and proprietary interfaces). And so if the service requirements change from the packet-network, such change cannot be handled by the transport network unless the new requirements exactly match with one of the other pre-existing service specifications. If it does not, then manual-coordination is required between IP and transport teams to draw up a new service definition and possibly a new contract; and then to translate this new service to actual circuit characteristics, potentially all the lines-of-code shown in Fig. 3.14 may need to be touched. Clearly this hampers extensibility of services or network features.

Distributed Applications: Even if we assume that the philosophy behind the UNI interface were to change and full visibility were allowed across the interface; it still does not change the fact that introducing new network functionality is hard, due to the distributed nature of their implementation coupled with the lack of a network-API.

Consider the challenges outlined below in implementing the network capability from Sec. 3.1, despite the use of a hypothetical full-visibility UNI:

- The head-end of an MPLS tunnel would need to create traffic-type specific tunnels (aggregates/bundles) and route them differently using DS-TE. But DS-TE only allows reserving bandwidth for a single traffic type.
  - One option could be that generic tunnels (with AutoRoute tuned off) may be used with policy-routing via manual (or scripted) use of the CLI to create the policy.
  - Or DS-TE can be changed to allow creating traffic-type tunnels for more than one type. But this would involve changing the link-reservation mechanism in OSPF-TE. And it would still need PBR to actually get specific-traffic into the tunnels.

- Once the traffic reaches a router at the edge of the transport network, it needs to be mapped into circuits designed for the traffic. This router acts as the UNI client and needs to know the requirements of each traffic-type:
  - Either by configuration, which would be static;
  - Or the tunnel-head-end has to have some way to communicate with this router, which would mean a new protocol;
- Assuming that the UNI client router is aware of the service requirements, it needs full visibility into the transport network. This requires the following:
  - One choice is to eliminate the UNI and make the (former) UNI client router part of the OSPF-TE instance running in the transport network. This is possible but now the router runs two instances of OSPF-TE (one in the packet later and the other in the circuit layer), which increases load on the router CPU.
  - The other choice is to change the UNI and convey summary information of the transport network to the router.

  In either case the UNI client would need to deal with conflicts in its decisions for services from the transport network with decisions made by other UNI-clients. It also needs to deal with possibly conflicting requests from multiple MPLS tunnel head-ends at the same time.
- Also monitoring of these circuits can by the management systems (NMS/EMS) today and theoretically it is possible to tie this monitoring capability to the CSPF algorithm running in the UNI client router, to re-route bandwidth based on usage. This would require change to the UNI or the use of a new protocol.
- Likewise such tie-ins can also be created with glue-code to develop a programmatic way for alarm generation (when network-failure events happen) to trigger a CSPF calculation and subsequent prioritized protection/re-routing. Again separate protocols and APIs would have to be used.

From the above discussion we see that full visibility across packets and circuits is essential to replicate our control-function, but even with full-visibility, a host of other glue-code and patchwork is needed to existing protocols, together with possibly some new protocols. And so it is easy to see that the current industry solution lacks ease of extensibility.

Common-Map Abstraction: Our SDN based solution is more easily extensible to the creation of new services or network functions (Fig. 3.15) because it gives full visibility of both networks to the network application in a programmatic way. It does not involve configured, presupposed/pre-baked service choices, mappings and CLI based policy rule-insertions. All of these can be dynamic and programmatic. And the common-map abstraction abstracts away the distribution mechanism from the applications, allowing them to be implemented in a centralized way with a well-defined and rich network-API.

For these reasons we believe the common-map abstraction, instead of the UNI, is the right abstraction for implementing simple and extensible services over packets and circuits. In the next section we give further validation of the extensibility of our architectural solution.

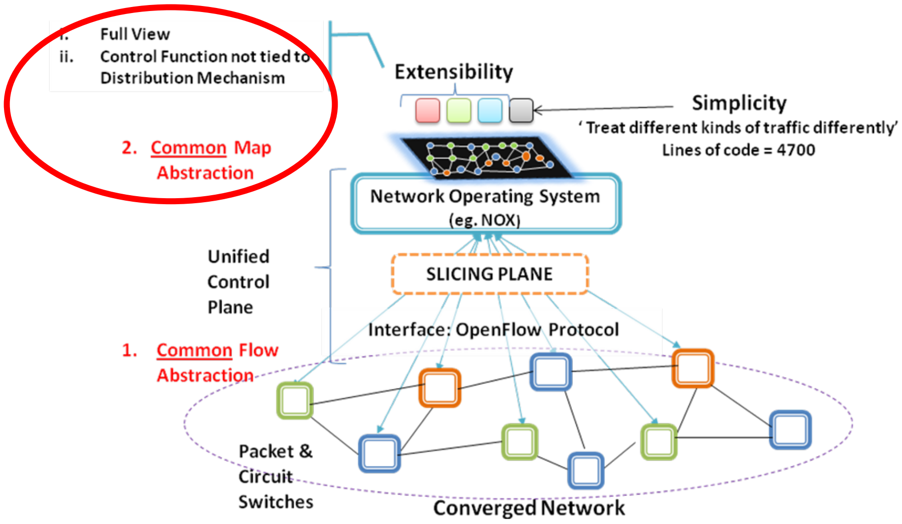

i. Full View
ii. Control Function not tied to Distribution Mechanism
2. Common Map Abstraction
Extensibility
Simplicity
'Treat different kinds of traffic differently'
Lines of code = 4700
Network Operating System (eg. NOX)
Unified Control Plane
SLICING PLANE
Interface: OpenFlow Protocol
1. Common Flow Abstraction
Packet & Circuit Switches
Converged Network

Figure 3.15: Simplicity & Extensibility of Unified Control Architecture

3.3.2 Other Applications with Packets and Circuits

In Section 3.1, we showed an example of our control architecture enabling network-capabilities that are end-user *application or service aware*. Such end-user services can be related to the traffic-type (voice, video, http, ftp) or some higher order application (like data-backup, live-TV, mobility). The network-capabilities for such end-user services include application-aware-routing, recovery and service-guarantees; all which can be easily performed with dynamic circuit-flows working closely with packet-flows. In this section, we briefly detail three other capabilities across packets and circuits made possible by our control architecture.

Dynamic Packet Links: IP topologies today are static. IP links over the wide-are pre-planned and put in-place using static-circuits in the underlying transport network. Information about link-state is carried by distributed link-state routing protocols to every router in the network. Routers make forwarding decisions based on SPF calculations performed on the map they create from the link-state information. Any change to the network-topology (say from link-failures) results in routing-protocol re-convergence, where every router re-converges to a consistent view of the network. Convergence is considered disruptive. Disappearing or re-appearing links require SPF calculations in every router, potentially re-routing packet flows everywhere in the network. Thus it is not surprising that IP links are always static.

But an OpenFlow based network eliminates the need for distributed routing protocols within the Controller's domain, as the switches do not make routing decisions. There are two advantages here:

1. With centralized-decision making, dynamic packet-link creation is convergence-free. We can create packet-links where none existed (using underlying dynamic circuits). And such newly created links no longer require distributed-routing protocol convergence, as routers no longer make routing decisions. And so circuits can change

as dynamically as needed to bring up/down packet-links in an IP-topology without fearing routing-protocol meltdown.

2. We can *choose* which flows to effect and which flow routes to leave unchanged. In sharp contrast to today's IP networks, none of these dynamic link-changes are disruptive to existing packet flows elsewhere in the network. The controller makes the decision of creating/changing/ deleting a link, and it only affects the flows chosen for the link and nowhere else.

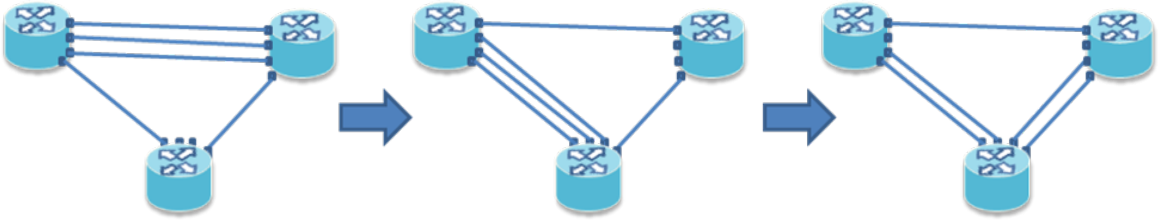


Figure 3.16: Dynamic Packet-Links

Fig. 3.16 shows how packet-links can be dynamically created and removed by redirecting underlying circuits (not shown) between three packet-switches, using the same set of packet-interfaces on each switch. The capability to create new packet-links can have several use-cases. One could involve relieving congestion between two routers. Other cases could involve time-of-day or application needs; where say between two data-centers depending on the time-of-day, or some bandwidth hungry application like data-backup, there is a need for more bandwidth. But at other times those same circuits could be re-routed away to buttress bandwidth needs elsewhere. This way expensive WAN bandwidth is shared between router-ports; bandwidth can be flexibly available where and when needed; and all of this can be done programmatically.

Variable Bandwidth Packet Links: There are two ways to consider packet-links that have variable bandwidth. One of these, shown in Fig. 3.17, is a variation of the dynamic links scheme proposed earlier. Here instead of using circuits to create *new* packet-links, *existing* packet links have their underlying circuit-bandwidth re-directed over the WAN.

Consider the 10GE router links in Fig. 3.17a. Instead of being supported by 10Gbps circuits in the transport network, one of them is supported by less (say 5 Gbps), while the other is supported at full line-rate. The lower part of 3.17a shows the same routers but also shows the transport equipment they are connected to. So when the lower-provisioned router-interface needs more bandwidth, circuits could be re-routed or created to support it. As an example of the former, in Fig. 3.17b, circuit-bandwidth has been re-routed from the previously fully provisioned router-link. Alternately, circuits could be created from elsewhere to support both router interfaces at full line rate, but only for a period of time; this is essentially a pay-per-use model for expensive WAN bandwidth.

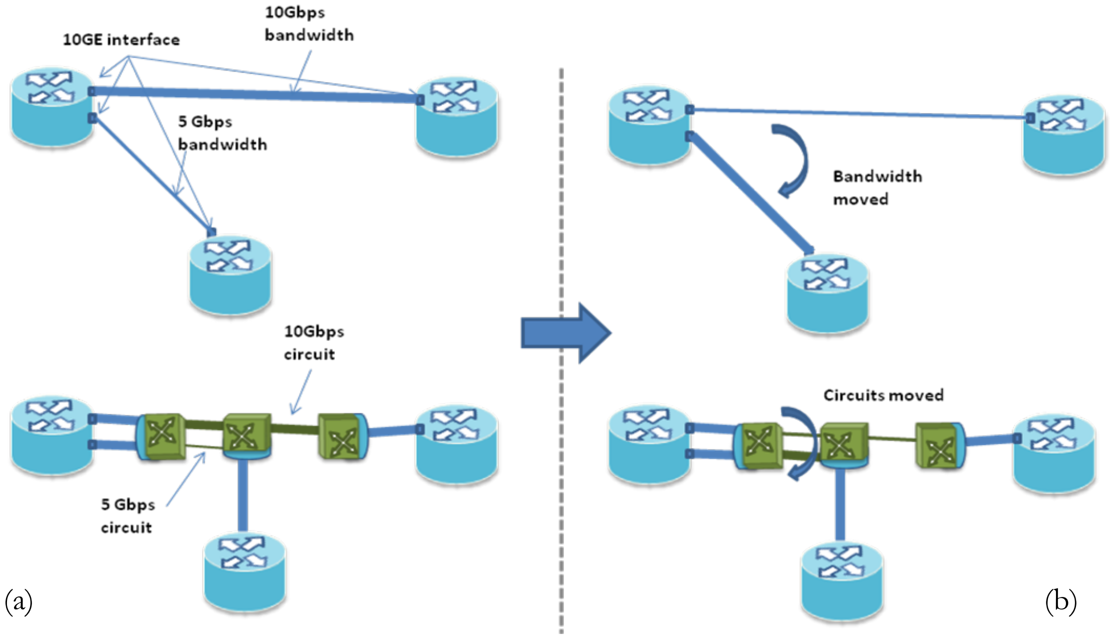


Figure 3.17: Variable Bandwidth Packet Links – Type I

The other kind of variable bandwidth packet-link involves dividing a packet link over multiple circuits of variable bandwidth in the transport layer. In Fig. 3.18, the same (virtual) packet-link uses two circuits (of variable bandwidth) to make up the full-line-rate of the router interfaces.

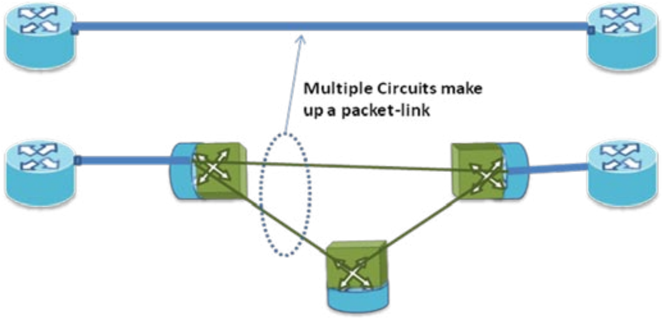


Figure 3.18: Variable Bandwidth Packet Links – Type II

Use-cases for doing so may include:

- load-balancing over different paths in the transport network;
- routing multiple circuits over diverse paths to support the same packet-link, so that the packet-link never goes down – if one of the circuit paths fail the other takes over with full line rate;
- Or as we showed in Sec 3.1.2, different services or traffic can be supported by different circuits (with different characteristics) over the same packet-link (i.e the same packet-interfaces).

Unified Routing: This particular application takes a look at how routing can be performed, were the packet and circuit switched networks considered part of the same 'layer'. In other words if the common-map were to be constructed based on the right-side of Fig. 2.12. Note that in this *de-layered* view, packet and circuit switches appear as a sea of switches connected by physical links (no virtual-links).

We show a way to do simple shortest-path IP routing; that does *not* resort to any complicated CSPF algorithms or traffic-engineering mechanisms; and yet, is aware of and accounts for network state and resources. Consider Fig. 3.19 - three packet switches (P) interconnected by *existing circuit flows* established on the circuit switches (C). Contrary to IP routing today, the connections between the packet-switches have costs assigned to them in a *piecewise* manner.

As an example, the circuits have costs inversely proportional to the bandwidth of the path, and directly proportional to the propagation delay and *usage* of the path. This way, higher the bandwidth of the circuit, or shorter the path, the cost of the circuit-path is lower. These parameters are fixed. However the usage parameter is variable and dependent on the actual traffic traversing the circuit flow. Higher the usage: the greater is the cost. As new packet flows get added to circuit flows, the usage (and cost) increases; discouraging the use of the circuit and potentially avoiding future congestion by making alternate (less-utilized) paths more attractive. As packet flows expire and usage goes down, the cost decreases.

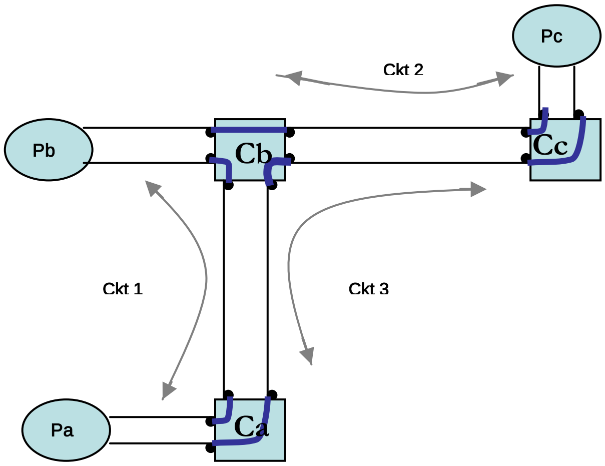


Figure 3.19: Unified Routing in a De-layered Packet-Circuit Network

The C to P connections also have costs assigned to them in a way that penalizes the use of the link. Let us see how – a packet-flow from Pa to Pc can take the direct path from Pa to Pc (via Ckt 3) or the indirect one that transits Pb. Assuming that the costs of the circuits sum up equally (ckt3 = cost of ckt1+cost of ckt2), the direct path would be preferred over the indirect path due to the penalty incurred in transiting Pb. So the packet flow gets routed over ckt3. But as circuit flow costs are variable, as usage increases on ckt. 3, the indirect path becomes more attractive.

To have 'usage' be reflected in the 'cost' used for an SPF calculation, allows efficient and yet simple routing. Unlike regular IP networks, the use of variable link-costs *do not* imply that packet flows already established in the network are effected (rerouted), as again the controller may re-route only the new flows entering the network, while leaving existing flows untouched.

To summarize, we provided further validation of the extensibility of our control architecture by showing examples of how packets can interact with dynamic circuits to provide varied services, beyond the ones discussed in Sec. 3.1. It is worth noting that most likely these services cannot be implemented with existing industry solutions due to lack of visibility between packets and circuit domains. But even if these examples could be implemented with existing industry-based control solutions, the sheer number of protocols, their distributed nature, and their potentially dangerous interactions, make the solutions so complex that no service provider would want to use them in their network.

3.4 Deployment Challenges & Proposed Solutions

As final validation of the simplicity and extensibility of our work, we propose solutions based on our control architecture to three deployment challenges *faced by any* unified control solution for packet and circuit networks[†].

Challenge #1: Reluctance of network operators of packet and circuit networks to share information with each other. We noted that UNI interface blocks all visibility at the boundary between IP and transport networks. The interface merely implements the *philosophy* of not wanting to share information across network boundaries. This remains an issue even with our solution (presented thus far). The operators of these networks simply *do not wish to share information* about their networks. And this problem is more acute in the case where the networks are owned by different businesses (analogous to AS-to-AS communications in the Internet where internal AS topology is not shared across AS boundaries). How then to create a common-map across packets and circuits?

Proposed Solution – Slicing & Transport Switching-as-a-Service: With reference to Fig. 3.1, so far we have only talked about the Net OS and OpenFlow as

[†] As an aside, note that the common-map abstraction for packet and circuit switching, applied to other network contexts such as data-centers and enterprise networks may *not* have these deployment challenges.

part of the unified control plane (UCP) that supports our abstractions. We now present the slicing-plane as a *key* component of the UCP that helps us meet several challenges.

In slicing, a new control layer is placed between the underlying switches and the controller-layer. This new control layer (the slicing layer) partitions the underlying data-plane into multiple slices, where each slice can then be put under the control of different controllers. An example of the implementation of the slicing-plane is the FlowVisor [50].

In the context of packet and circuit networks, a slice is defined as the combination of bandwidth *and* circuit-switching resources. It is worth noting that transport networks today also provide slices of their network to ISPs. However such slices only provide *static* bandwidth with no control over that bandwidth (where by control we mean the ability to route it differently, change its size etc.). Such static bandwidth is often referred to as a dumb-pipe in the industry.

However our slice definition includes *bandwidth plus the ability to manipulate it* via control over switching resources in that slice [11]. Circuits in a slice already provide data-plane isolation between slices. The slicing plane crucially provides *control-plane* isolation. With the slicing plane under the transport network operator's control, the control of each slice can be turned over to the ISP that purchases the slice (Fig 3.14).

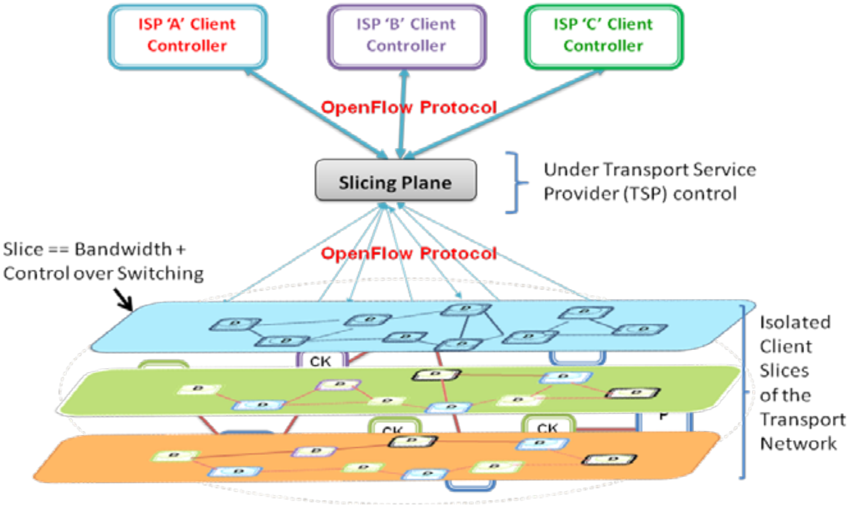

ISP 'A' Client Controller
ISP 'B' Client Controller
ISP 'C' Client Controller
OpenFlow Protocol
Slicing Plane
Under Transport Service Provider (TSP) control
Slice == Bandwidth + Control over Switching
OpenFlow Protocol
CK
CK
CK
Isolated Client Slices of the Transport Network

Figure 3.19: Slicing the Transport Network [11]

The slicing-plane ensures that no-ISP can maliciously or inadvertently control any transport switching resources that are not in its slice of the transport network. The slicing plane achieves this control-isolation by monitoring the messages between the switches and the controllers and checking them against the slice-policy. It can either passively relay the messages or translate them where appropriate. Or it can reject the messages if they are in violation of the slice-policy.

Slicing enables the ISP to run the common-map abstraction in its controller. This controller has view of the packet switches that belong to the ISP. It also has view of the circuit-switching-resources that are part of its slice of the transport network, thereby enabling the ISP to create a common-map abstraction across packets and circuits. In other words the 'slice' *is* the common-map that presents the same map-abstraction and network-API discussed in Chapters 1 and 2.

Note that this view of the circuit-switching-resources in the transport network is restricted to *only* the slice that the ISP has purchased *and not* the entire transport network. It thus overcomes the lack-of-information-sharing seen in packet and circuit networks today, by sharing only a *part* of the information (instead of divulging information about the entire-network). And the transport network carrier has *incentive* to share this partial information about the slice because it enables a new service (new revenue opportunities). Today transport networks share no information and offer only static-bandwidth (dumb pipes). With slicing, transport networks can offer slices which include bandwidth *and* isolated transport-switch control to ISPs, thereby offering a new service [73] [†].

It is worth noting that in the absence of such sharing models, the only option is to share *no* information, leading to interfaces between IP and transport networks like the UNI. And we have shown issues with such interfaces in the previous sections, which we believe are key reasons behind the commercial-failure of the UNI (and GMPLS). But with our proposed *technical* solution based on slicing, and our *economic* solution of offering transport slices-as-a-service, we believe the practical-challenge of information sharing can be solved.

[†] The creation of the slice or modifications to what falls within a slice requires an out-of-band channel (either manual or automated), and is outside the scope of OpenFlow. Once the slice is created, our control architecture provides control over switching resources in the slice and control-plane isolation between multiple slices.

Challenge #2: Conservative nature of transport network operators towards automated control planes. Transport network operators would like to respond faster and provide more dynamic services to meet their client needs, but loathe giving up precise *manual control* over the way traffic is routed over their network to a software control plane, irrespective of how intelligent that control plane may be [26]. Is there a way that allows decades of established procedures to co-exist with new ways to deploy services and operate the network?

Proposed Solution – Slicing based Gradual Adoption Path: Again it is the slicing plane that provides a gradual adoption path, that can let established procedures co-exist with new ones. Consider this – the transport network operator initially slices only a small part (5%) of its network and hands over control to an ISP's controller through its slicing plane. As long as the slicing plane guarantees isolation of the slices in both the data and control plane, the transport network operator *retains* control over the unsliced part of the transport network which can still be run manually, using established procedures, as it is today. Meanwhile the ISP is free to use whatever automated intelligent control algorithms it may desire in its isolated slice of the transport network (any of the examples from the previous section). Over time as more confidence is gained in the slicing framework, more parts of the transport network could be sliced and offered as a service to other ISPs. It is worth noting that GMPLS provides no such means for such flexible and gradual adoption.

Challenge #3: A final challenge faced by any unified control plane solution for packet and circuit networks, involves the conservative nature of IP network operators towards increasing load on distributed routing protocols.

When the *same* business owns both networks, a *single* instance of a distributed routing protocol could disseminate information on packet and circuit link state leading to the creation of a common-map. However this is not done today simply because IP network operators loathe exposing their distributed link-state routing protocols to transport network link attributes in fear of increasing load on router-CPUs and de-stabilizing their network. Let’s see why.

Distributed link-state routing protocol like OSPF or IS-IS have convergence and stability issues if network state changes too fast or too often. Stability in a distributed routing protocol can be defined in several ways:

- A clear indicator is the network convergence time i.e. the times taken by all the routers to return to steady state operation after a change in network state. Convergence time has a propagation component, which is the time taken for the new information to be disbursed via the IGP to all the routers so they can update their link-state databases; and a re-route component where new SPTs are calculated or TE paths are re-routed if necessary. Clearly a low convergence time indicates a stable network.
- A related parameter often is the number of route flaps, which refer to routing table changes in a router in response to change in network state. A large number of flaps in a short time can adversely affect network stability.
- But by far the key indicator of stability is the routing load on switch CPUs. This is a measure of how much time a router spends in processing control packets from the routing protocol. If there are frequent changes in the network state, and update messages are sent frequently (sometimes called churn), the router CPU may spend a lot of time doing route calculations; this in turn may lead to ingress CPU queues getting filled and incoming packets getting dropped (including hellos); dropping keep-alives may cause timeouts and dropping of routing- adjacencies; which again leads to more control packets getting generated and greater routing load. This cascading effect results in longer convergence times, routing loops, more route flaps, greater CPU loads, and in the worst case network meltdown [74].

To avoid routing protocol instability, router vendors apply several damping mechanisms to keep things under control. For example, the router is informed of bad news (link down) by the link layer fast, but good news (link up) is delayed (many seconds) to avoid potential link flaps. The generation of link update messages and the occurrence of SPT recalculations are throttled by dynamic timers that exponentially increase their times during periods of instability to prevent processor meltdown.

It is therefore easy to see that distributed routing protocols are fragile, and require careful tuning and tweaking to keep them stable. Changes that are too fast or too often are not tolerated and carefully damped. This is the fundamental reason why IP networks today do not support dynamic links or dynamic link weights [75].

And so to extend OSPF/IS-IS and use it in a dynamic circuit network *with its effect being felt* by the same or another instance of the distributed routing protocol in the packet network is simply dangerous and unwarranted. Unfortunately, this is exactly what GMPLS proposes.

Our Solution – Elimination of Distributed Routing Protocols: We have mentioned several times before that distributed routing protocols are unnecessary in an SDN controller's domain. The elimination of distributed routing protocols is a natural consequence of the adoption of SDN ideas and OpenFlow to form our unified control architecture. But it is worth pointing out that in the context of packet and circuit network control, we find that SDN/OpenFlow has the direct benefit of removing one of the *key hindrances* to the deployment of dynamic circuit switching in today's networks; by no longer being subject to a) stability issues of dynamic routing protocols, and b) limited processing powers on switch CPUs.

3.5 Summary

The goal of this chapter was to validate the simplicity and extensibility claims we made for our architectural solution. For the simplicity claim, we took a new and fairly involved network capability; implemented it with our control architecture; and then compared our implementation to our best guess of what it would take to implement the same functions using current industry-standard solutions for packet-circuit control.

We performed a lines-of-code analysis and found our solution to be *two* orders-of-magnitude simpler. We provide qualitative architectural insights into why our solution takes far less line-of-code. And while our work is not production-ready, two orders of magnitude difference gives us confidence that even production quality code in our architecture would be much simpler.

Next we reasoned that even with two orders of magnitude more code, current industry solutions would not be able to exactly replicate our work. We believe it is because of two main reasons: the use of the UNI interface which results in loss of visibility across packet and circuit networks; and the implementation of services as distributed systems which are tied to the distribution mechanisms. We explained how our common-map abstraction does not suffer from these limitations; and therefore not only is it simple to implement control-functions across packet and circuits; it is easy to introduce new functionality, or change existing functionality in the network just as easily; thereby validating our extensibility claim. As further justification of extensibility, we described three other applications enabled at the packet-circuit boundary by our unified-control architecture.

Finally we detailed deployment issues faced by *any* control-solution for common packet-circuit control. We proposed the use of ‘slicing’, a key component of our control architecture, to overcome these challenges. For transport network operators, slicing is a technical solution that a) provides an economic-incentive (a new service) for sharing information with ISPs, which can then allow the latter to have visibility across packets and circuits when creating the common-map; and b) slicing eases the path to gradual-adoption of our control architecture – a key requirement for the adoption of any new control technology.

Chapter 4

Network Design & Analysis

In the introductory chapter, we made the claim (in Sec. 1.3) that circuit switching was here to stay in the core, as it can make the Internet more efficient, if packet and circuit networks work together.

Accordingly, we proposed packet-and-circuit network convergence (pac.c network); where packet-switched IP networks and dynamically-circuit-switched transport networks work together under a common control-plane based on an SDN approach. And we showed how our control architecture affords simplicity and extensibility in providing network functions and services, when compared to existing industry-standard solutions for packet and circuit network control.

The goal of this chapter is to validate the *efficiency* claim. We investigate the Capex and Opex implications and savings afforded by convergence with our control-architecture when compared to current industry practices.

The networks that interest us in this chapter are primarily IP networks. Even though we model the transport network in each case (point-to-point WDM, optical bypass and dynamic-circuit-switching), we do so *only in support of the IP network*. Another way to think about this approach is the one shown in Fig. 4.1a (Ch 1 – Fig. 1.6a), where in one possible future, all services are provided on top of the Internet, and the transport network exists *only* to support *one or more* IP networks.

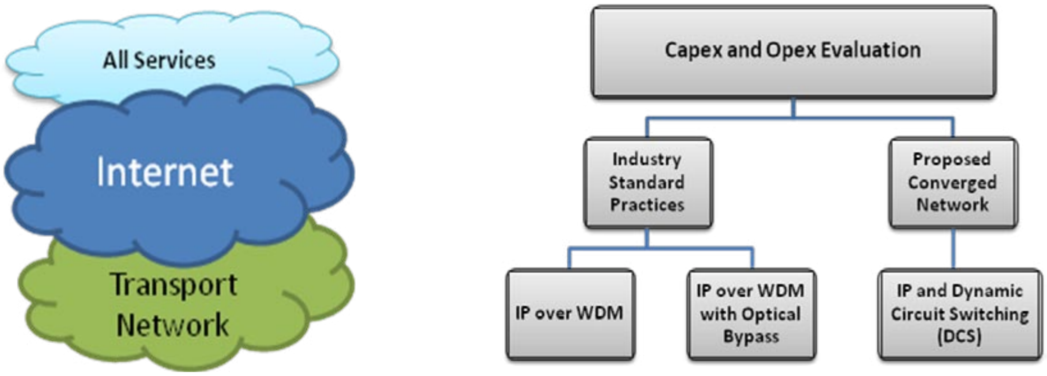

All Services
Internet
Transport Network
Capex and Opex Evaluation
Industry Standard Practices
Proposed Converged Network
IP over WDM
IP over WDM with Optical Bypass
IP and Dynamic Circuit Switching (DCS)

Figure 4.1: (a) One possible future (b) Evaluation Procedure

Our approach to Capex and Opex analysis is as follows (Fig. 4.1b):

1. We first outline a design methodology for a core IP network that is completely packet-switched – i.e. all switching is performed by IP routers; and these routers are connected over the wide-area by point-to-point WDM line systems. In other words, the transport network does not use any circuit switches to support the router-links and is completely static. This is in-fact the IP-over-WDM design scenario presented in Sec. 1.3, and is a popular way for constructing core-IP networks today. We will consider this as the reference design and model it in Sec 4.1.1.
2. Next we consider a small variation of the reference design by adding optical-bypass. This is a technique by which the number of required core-router ports is reduced by keeping transit-traffic in the optical domain (typically at a wavelength granularity). Optical bypass or express-links are used today in core-IP networks and also discussed in literature [79, 82, 83]. But it's important to note that while optical-bypass can be achieved with optical *switches*, it is nevertheless a *static* approach – the optical switches do not switch wavelengths dynamically. IP over WDM with optical-bypass is modeled in Sec. 4.1.3.
3. The first two steps cover industry-standard practices. In the final step we consider our proposed network that uses both packet-switching and dynamic-circuit-switching (DCS) under a common control plane. While DCS can be used with IP in varying

degrees, we model an IP-and-DCS network with the following three characteristics: a) we replace Backbone Routers in PoPs with Backbone Packet-Optical Switches; b) we use a full-mesh of variable-bandwidth circuits between core-PoPs; and c) we adopt our unified control architecture[^] for common control of packet and circuit switching. In Sec. 4.2.2 we discuss these design choices and the motivation behind them in terms of the benefits they afford when compared to the reference design; and we outline the design methodology for such a converged network in Sec. 4.2.3.

Analysis of our converged network (in Sec. 4.2.4) shows that we can achieve nearly 60% lower Capex costs for switching hardware compared to the IP-over-WDM reference design; and 50% lower Capex compared to IP-over-WDM with static optical-bypass. Importantly, we show that while the savings achieved by optical-bypass (10-15%) can get eliminated if we vary the traffic-matrix, Capex savings in a pac.c network are insensitive to varying traffic-matrices. Additionally, our design scales better (at a lower $/Tbps slope) when we scale the aggregate-traffic-demand from 1X to 5X. In Sec. 4.2.5, a limited Opex analysis that considers power-consumption, equipment-rack-rentals and network-technician man-hours, shows nearly 40% in savings compared to the reference design.

4.1 Reference Design: IP over WDM

The core-IP network designed with an IP-over-WDM[†] approach can be modeled as a collection of routers in different city PoPs (Points-of-Presence). The backbone routers in these PoPs are interconnected over the wide area by ‘waves’ (industry parlance) leased from the transport network. Such waves are manifested by wavelengths stitched together in point-to-point WDM line-systems. There is no circuit-switching (static or dynamic) in the transport network. In fact, in most cases, circuit-switches are not used at all to provision these waves (which is what we model). As mentioned before, our focus is on

[^] Architectural details were covered in Ch. 1- Sec. 1.4 and Ch 2 – Sec. 2.1 and 2.2.

[†] Unfortunately, such design is known by many names in the literature – IP over Optical [77], pure IP [79], IP over DWDM with TXP [83]. Others thankfully call it IP-over-WDM [84]

the IP network; we do not model the transport network individually; nor do we model the entire transport network[^]; and we do not consider *other* services or networks the transport network might support today.

It is worth noting that while our design methodology is detailed and comprehensive in its steps, it does *not involve any optimization*. No attempt has been made to optimize the design for any particular design criteria[†], because optimization is not the goal here. Instead we wish to obtain ballpark-numbers for the relative comparison. As a result the IP network is designed as a pure IP network sans any use of MPLS based traffic-engineered tunnels.

4.1.1 Design Methodology

The design methodology for IP-over-WDM design is briefly outlined below:

1. Network Topologies: We choose representative topologies for both IP and WDM networks for a large US carrier like AT&T.
2. Traffic-Matrix: We create a unidirectional (IP) traffic matrix from all cities in the IP topology to all other cities in the topology. Each element in this matrix represents the average-traffic (in Gbps) sourced by one city and destined for the receiving-city.
3. IP Edge-Dimensioning: In this step we account for all the traffic that could traverse an *edge* in the IP topology. Such traffic includes a) the average traffic-demand between cities routed over the edge; b) traffic-rerouted over the edge in the event of failures; and c) head-room (over-provisioning) for variability in the traffic-volume in the previous cases.
4. IP PoP-Dimensioning: Once all the edges in the IP topology have been dimensioned, we calculate the number of parallel 10G *links* that make up the edge; and the number of Backbone and Access Routers required in a PoP to switch between those links.
5. WDM-Network Dimensioning: Finally each 10G link in the IP network is translated to a wavelength path in the WDM network. The path is determined by routing the

[^] For example, the transport network may be substantially bigger in terms of the number of switching nodes, compared to the IP network it supports (as highlighted in [77]).

[†] Such design criteria could include the optimal routing of IP links to minimize concurrent failures, or the use of diverse-paths for routing, or the minimization of transponders in the transport network, etc.

(virtual) IP link over the WDM (fiber) topology. Once all IP links have been routed, we calculate the number of WDM line-systems, optical components and WDM transponders required to satisfy the demand.

Design Steps: We give more details and partial results for each step as follows:

1. Network Topologies:
   a. We use AT&T's IP network as reported by the Rocketfuel project (Fig. 4.2a) [78][†]. The Rocketfuel topology gives us node-locations (cities) and edges (inter-city connections). On this we layer typical PoP structure of access-routers (AR) dual-homed to two backbone routers (BR) (Fig. 4.2b). The ARs are either local (situated in same city PoP as the BRs) or housed in remote-cities [77]. The ARs aggregate traffic from the local and remote cities into the BRs, which are responsible for switching traffic to other core-PoPs over the backbone edges. This structure results in the hub-and-spoke look of Fig.4.2c; where the hubs are the core city-PoPs (in 16 major cities); the spokes represent remote-sites (89 cities) that use remote-ARs to connect to the core-city PoP's BRs; and 34 backbone edges that connect the BRs in the PoPs over the wide-area. Our use of the term *edge* to represent the wide-area inter-city connections will be resolved in the dimensioning process, into multiple parallel 10 Gbps IP *links*.

   b. For the WDM network, we use a topology from [79] shown in Fig.4.3a. Although [79] does not give details of the node-locations, we layer the topology on a map of all North-American fiber routes [80] (Fig. 4.3b), to get the node locations and link-distances. The fiber-topology includes 60 nodes and 77 edges, where the longest edge-length is 1500 km and the average link length is 417km. The physical-edges of the WDM topology will be resolved in the dimensioning process into multiple parallel fibers and 40-wavelength C-band WDM line-systems.

[†] Note that network details (topology, number of switches, etc) are closely guarded secrets which carriers never divulge. The Rocketfuel project uses clever mechanisms to trace out network-topologies for several large ISPs.

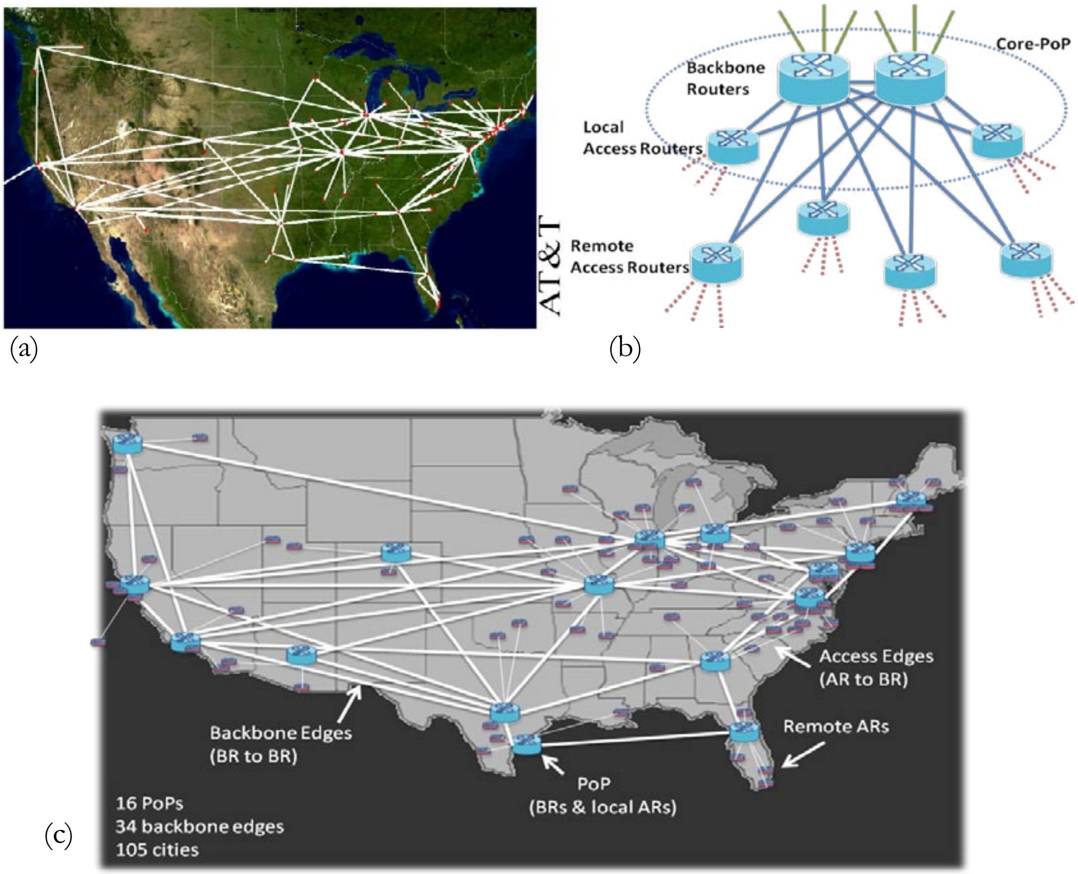


Figure 4.2: (a) AT&T's IP network [78] (b) PoP structure (c) IP topology

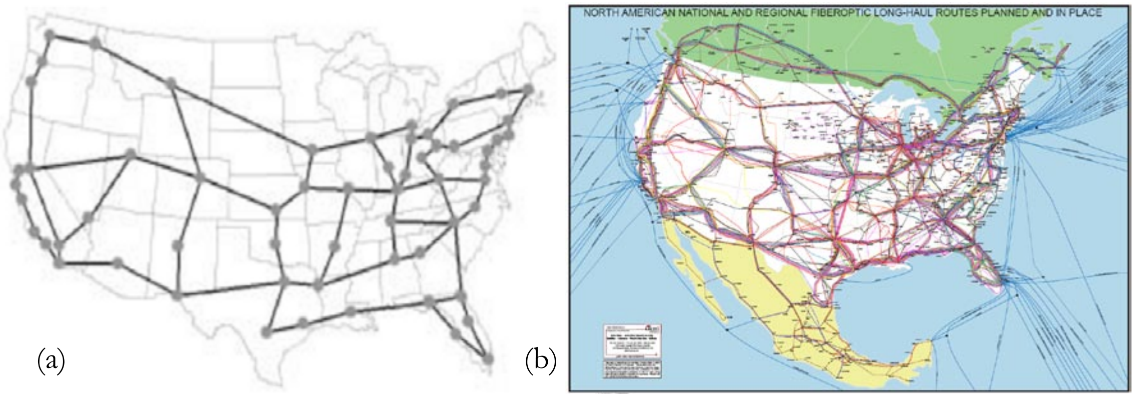

(a)
(b)
NORTH AMERICAN NATIONAL AND REGIONAL FIBEROPTIC LONG-HAUL ROUTES PLANNED AND IN PLACE

Figure 4.3: (a) Fiber (WDM) topology [79] (b) North-American Fiber Routes [80]

2. Unidirectional Traffic Matrix: We use a gravity-model as a starting point for a traffic matrix [81]. For each of the 105 cities in the IP topology, we estimate the traffic sourced to the other 104 cities, by multiplying their populations and dividing by some power of the physical-distance between them. Note that the power can be zero as well, in which case distance is no longer a factor. We then scale the traffic-matrix entries to achieve a cumulative traffic-demand on the IP network of 2 Tbps[†]. Note that this traffic matrix only considers the ISP's internal traffic. It does not consider the traffic to and from other-ISPs with which our ISP peers. Additionally, estimating or predicting traffic matrices correctly is hard even for ISPs. Thus, later in the analysis we vary the traffic matrix to study the effects on network Capex. We will also scale the traffic matrix from 2X to 5X times the original aggregate demand, to study resultant effect on Capex and Opex.
3. IP Edge-Dimensioning: This step is at the heart of the design process for the IP network and is actually a combination of several steps:
   a. First the traffic-matrix is dimensioned on the core IP-network. Every demand in the IP traffic-matrix is routed from source-AR to destination-AR over the backbone IP topology (Fig. 4.2c). The route from source-AR to dest-AR is based on Dijkstra's shortest-path-first (SPF) algorithm, with the following assumptions:
      - Load-balancing (via ECMP) or traffic-engineering mechanisms (MPLS-TE) are not used.
      - The metric used in the SPF algorithm is hop-count[*].

      The demand traffic is accounted for on each edge in the resultant shortest-path route. For example, in Fig. 4.4a, demand-traffic from AR1 to AR2, is routed via BR1-BR3-BR2. Note that BR3 'sees' this traffic as 'transit' traffic. The demand AR1→AR2 is tabulated on the core-edges between BR1→BR3 and BR3→BR2, and the access edges AR1→BR1 and AR2→BR2, accounting for direction of traffic flow.

[†] From our discussion with ISPs, 2Tbps is a reasonable estimate of aggregate traffic demand on a US continental ISP's core network, for fairly large carriers at the time of this writing (2011).

[*] This is realistic – in the absence of any special metrics set by the network-operator for an edge in the IP topology, routing protocols such as OSPF and IS-IS default to the same number for the routing metric for each edge, effectively reducing the SPF calculation to a shortest-hop-count calculation. The metric for each edge is set by the network operator as an optimization, and as mentioned in the introduction we are not interested in optimizing the design for any special purpose (which is also why we ignore ECMP and MPLS-TE)

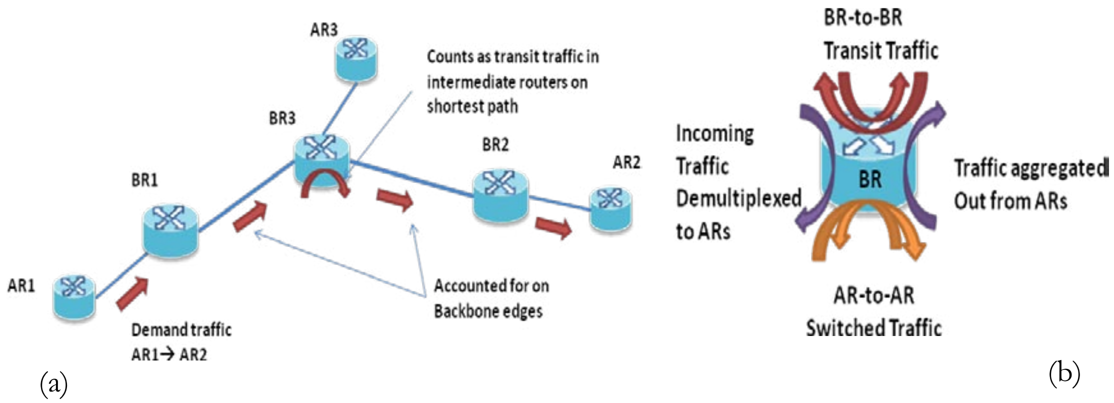

AR3
Counts as transit traffic in intermediate routers on shortest path
BR3
BR2
AR2
BR1
AR1
Accounted for on Backbone edges
Demand traffic AR1→ AR2
(a)
BR-to-BR Transit Traffic
Incoming Traffic Demultiplexed to ARs
BR
Traffic aggregated Out from ARs
AR-to-AR Switched Traffic
(b)

Figure 4.4: Dimensioning the IP network

We also keep track of the traffic ‘seen’ by the BRs in each PoP. From Fig. 4.4b, this includes the traffic switched locally between the ARs by the BRs; the traffic aggregated and ‘sourced-out’ by the BRs destined to other PoPs; the incoming-traffic from other PoPs destined to the ARs serviced by a PoP; and finally, the traffic that *transits* through the BRs in a PoP, on their path to a destination PoP.

b. Next, we account for failures by dimensioning for recovery. We break an edge in the IP backbone-topology and re-route the *entire* traffic-matrix over the resultant topology, which now has one less edge. Again this is precisely what would happen as a result of individual Dijkstra calculations in every router in the network. This time we get new values for the aggregate-traffic routed on each edge of the failure-topology. By breaking each edge of the topology one-at-a-time and tabulating the re-routed demands, we get different numbers for each edge for every link-failure scenario. We then repeat this process by breaking each node in the IP-core topology.

   - Assumption: We only consider single-failure scenarios – i.e if an edge in the IP topology breaks, no other IP edge or node breaks at the same time. If a node breaks, then transit through the node is not possible, but traffic can still be sourced in and out of the node due to dual-homing of ARs to BRs.

o Assumption: We will see shortly that each *edge* in the IP topology is actually instantiated by several parallel IP-*links*. We assume that the breakage of the edge corresponds to the breakage of all links that make up the edge.

Neither assumption mentioned above is entirely true in practice, and depends on factors such as the routing of the individual IP links over the underlying fiber network as well as the location and the cause of the failure. But to keep the analysis simple and fair, we make the above assumptions and keep it consistent across all design scenarios.

c. At the end of the previous step, for *each edge* in the IP topology, we have the following set of tabulated traffic in *each direction*– 1 for the original no-failure demand-matrix, 34 link-failures and 16 node-failures (the max. of these is show in Table 4.1 for a few edges in the IP topology). For each edge we pick the highest aggregate out of all these cases for each direction. Then we pick the higher value from the two directions, and set that as the bi-directional demand for each IP edge (last column in Table 4.1).

Bidirectional IP Edge	Demand -->	Demand <--	Recovery_max -->	Recovery_max <--	Edge Dimensioned
Chicago <=> NewYork	125.14	130.36	203.64	220.89	220.89
LosAngeles <=> Chicago	97.39	96.95	118.28	177.82	177.82
WashingtonDC <=> NewYork	97.19	90.53	133.31	124.79	133.31
Orlando <=> Atlanta	91.74	90.72	177.40	167.53	177.40
Atlanta <=> WashingtonDC	82.25	71.16	120.42	108.07	120.42

Table 4.1: Routed traffic for a few IP edges (all values in Gbps)

d. Finally we dimension for traffic variability by over-provisioning the edges. Such over-provisioning can be performed by dividing the bi-directional traffic demand from the previous step, by an allowable-link-utilization factor [77]. We chose a utilization factor of 25%, which translates into 4X over-provisioning.

e. So far we have accounted for all backbone-edges (BR to BR). To account for the access-edges (AR to BR) we use the max. of the aggregate traffic sourced or sunk by the access-city. We account for failures by doubling this demand, as access-links are dual-homed to backbone routers. Finally we use the same over-provisioning factor we use in the backbone edges and set the resultant value as the bidirectional traffic-demand for the access edge.

4. IP PoP-Dimensioning: Now that we have each edge of the IP network dimensioned for demand, failures and traffic-variability, we can figure out the number of routers (ARs and BRs) in the PoPs and the number of links (or ports) that make up each edge (Fig. 4.5).

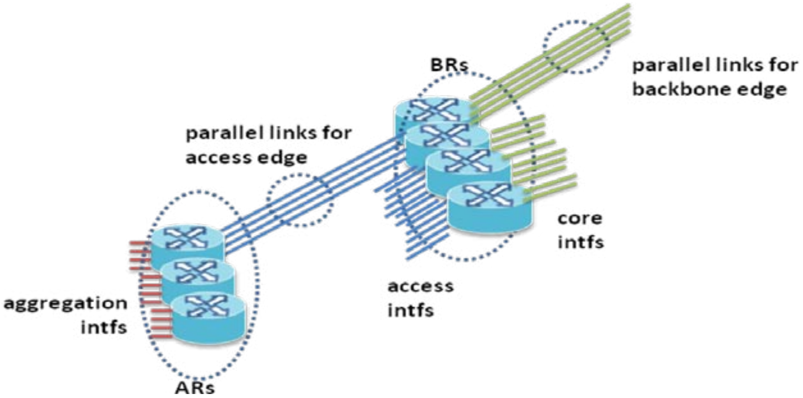

BRs
parallel links for backbone edge
parallel links for access edge
core intfs
access intfs
aggregation intfs
ARs

Figure 4.5: Determining the Number of Routers and Links

a. The number of parallel access and backbone links (or interfaces) can be determined by simply dividing the edge demand by the linerate of a single interface (assumed 10Gbps for all interfaces).

b. The number of access-routers in each access-city can be determined by summing up the number of access-to-BR interfaces, doubling it to account for aggregation interfaces (assumed equal to the access-interfaces), multiplying by the line-rate and dividing by the AR switching-capacity (assumed 640 Gbps).

c. The number of core-routers in each PoP is determined by summing up all the access and core interfaces, multiplying by the line-rate, and dividing by the switching-capacity of a single BR (assumed 1.28 Tbps).

City PoP	Number of Outgoing Parallel Links to BRs in other City PoPs
Seattle	Chicago: 21, LosAngeles: 24, SanFrancisco: 3
SanFrancisco	Chicago: 42, Dallas: 30, Denver: 2, LosAngeles: 34, StLouis: 9, Seattle:3
LosAngeles	Atlanta: 51, Chicago: 72, Dallas: 58, StLouis: 57, SanFrancisco: 34, Seattle: 24

Table 4.2: Number of parallel IP Links making up an IP Edge for a few PoPs

5. WDM Network-Dimensioning: In this step we route each IP-edge over the physical-fiber topology (Fig.4.6) to account for WDM network requirements.

   a. Again the IP edge is shortest-path routed over the fiber topology, but this time the metric used is the physical-distance of each edge in the fiber-topology (instead of hop-count). Assumption: no optimization for routing IP edges on phy topology.

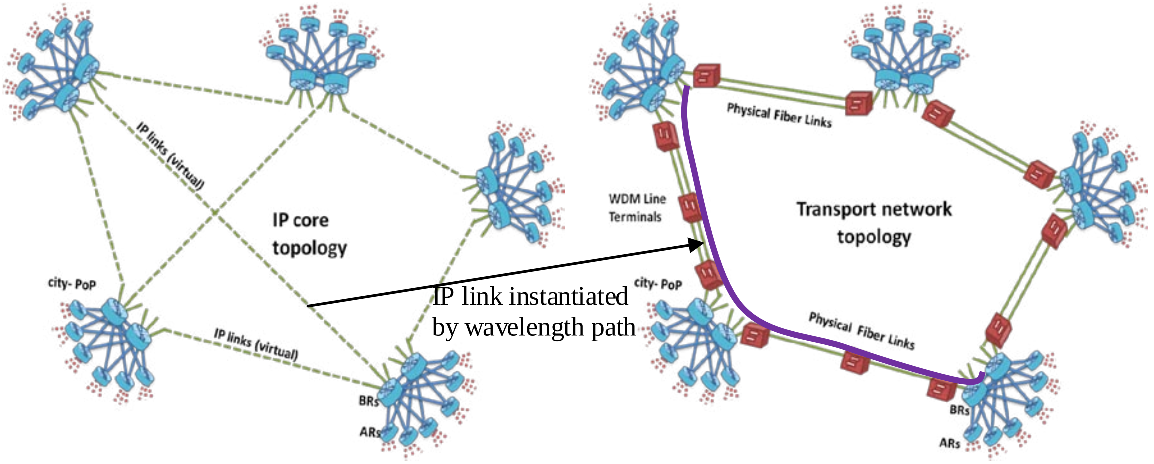


Figure 4.6: Routing an IP link on the Physical topology

   b. The number of 10G interfaces calculated for the IP edge in the previous step translates into the number of 10G waves demanded from the WDM network. This demand is then tabulated for each edge in the fiber topology over which the IP-edge is routed. Assumption: all links that make up the IP edge are routed the same way in the physical topology.

   c. Then on a per-physical-edge basis, we tabulate the aggregate 'waves' routed. From this we can figure out the number of *parallel* 40ch, C-band WDM line systems required. For example, in Table 4.3, if the demand on a physical-edge is for 326 waves, 9 parallel line-systems would be required, with 8 of them fully-lit and 1 partially lit (6 of the 40 waves will have transponders on either end).

d. We also account for the number of line-systems required *in series*, by noting the length of the physical-link and dividing it by the reach of the WDM line system. We assumed a line-system with 750km optical reach. Beyond 750km, the waves have to be regenerated electronically using back-to-back transponders.

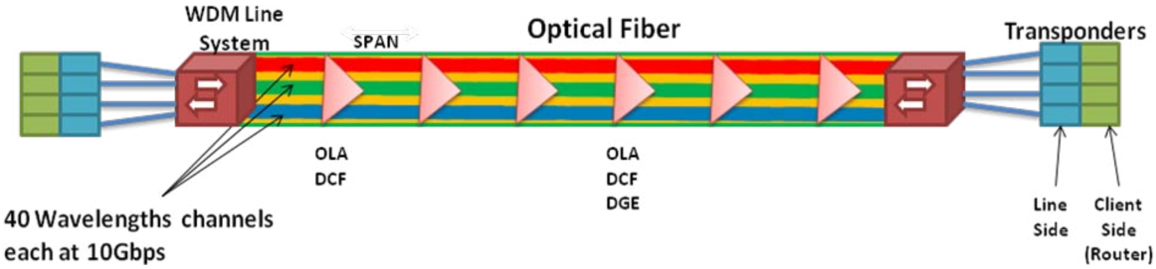


Figure 4.7: WDM Line System

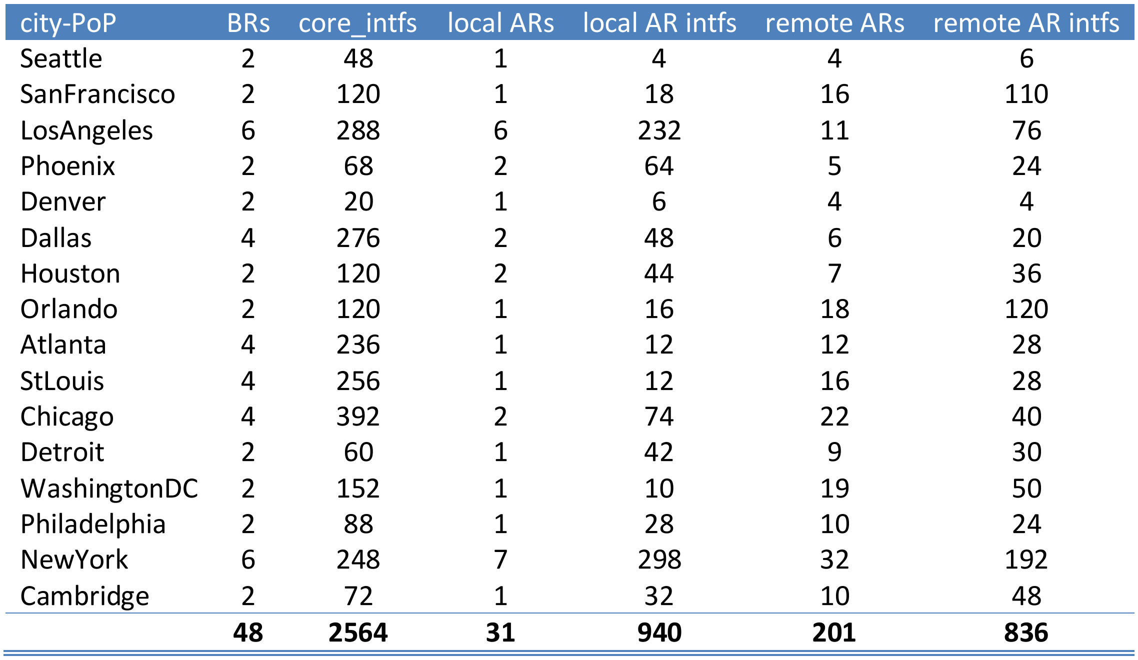
e. Finally, we account for the optical components used in the fully and partially lit systems (Fig. 4.7). These include WDM transponders with client and line-side transceivers – the client-side connects to the router interfaces with short-reach optics (< 2km) typically at the 1310nm wavelength; whereas the line-side contains long-reach optics at ITU grid wavelength (100s of km). The line-systems also contain wavelength multiplexers and de-multiplexers, pre/post and in-line amplifiers (OLAs with a span of 80km), dispersion compensators (DCFs co-located with each amplifier) and dynamic gain-equalizers (DGEs co-located every 4th amplifier). Table 4.3 shows results for a few edges in the fiber-topology.

Bidirectional Physical Edge	Wave Demand	Link length(km)	Parallel LineSys	lit-waves in last	Full-reach Set of LS	Dist. of last (km)
ElPaso <=> Dallas	326	1027	8	6	1	277
Sacramento <=> SanFrancisco	80	141	2	0	0	141
Nashville <=> Charlotte	8	658	0	8	0	658

Table 4.3: WDM Line Systems satisfying demands for few Edges in Physical Topology

4.1.2 Results and Capex Analysis

With the IP-over-WDM design methodology detailed in the previous section, PoP dimensioning results are shown in Table 4.4. It includes, for each PoP, the number of BRs, local-ARs and remote ARs; as well as the core-facing interfaces on the BRs; and the interfaces on the ARs that connect to the BRs. Naturally the access-facing interfaces on the BRs total to a sum of the access interfaces on the local and remote ARs.



city-PoP	BRs	core_intfs	local ARs	local AR intfs	remote ARs	remote AR intfs
Seattle	2	48	1	4	4	6
SanFrancisco	2	120	1	18	16	110
LosAngeles	6	288	6	232	11	76
Phoenix	2	68	2	64	5	24
Denver	2	20	1	6	4	4
Dallas	4	276	2	48	6	20
Houston	2	120	2	44	7	36
Orlando	2	120	1	16	18	120
Atlanta	4	236	1	12	12	28
StLouis	4	256	1	12	16	28
Chicago	4	392	2	74	22	40
Detroit	2	60	1	42	9	30
WashingtonDC	2	152	1	10	19	50
Philadelphia	2	88	1	28	10	24
NewYork	6	248	7	298	32	192
Cambridge	2	72	1	32	10	48
	48	**2564**	**31**	**940**	**201**	**836**

Table 4.4: IP network Router and Interface counts per PoP

From Table 4.4 we find that our IP network consists of 280 routers, 48 of which are BRs, and the rest ARs. The BRs collectively have 2564 core-facing interfaces. While studies on IP-over-WDM have been published in literature, there remains a lot of variability in the design process. Some studies ignore network-recovery [82]; others ignore differences in IP and WDM topologies [79]; and there isn't any use of standard topologies or traffic-matrices [77, 79, 82-84]. All of these factors can result in fairly different numbers when tabulated for router /port-counts, making comparisons between studies hard.

Additionally most papers do not even report router and interface counts and just state Capex results. Of the few works that do list port-counts, consider recovery and over-provisioning, [79] states (in Table 8.7) that a 1.7 Tbps IP network would use 3479 core-ports, which is in the same ballpark as our tabulation of 2564 ports in a 2 Tbps IP network. The higher count in [79] could possibly be a result of the (curious) use of identical IP and WDM topologies. Also from our discussions with ISPs, core networks with a few hundred routers are typical. If anything, our work is a lower-end estimate of router and port counts, due to our somewhat simplified PoP structure (compare Fig. 4.2b to Fig. 3 in [99]).

From Fig. 4.8a, we get a relative idea of the inter-PoP (not inter-AR) traffic-matrix. The figure shows the traffic collectively-sourced (or originated) by the ARs in the PoPs to other parts of the network. This figure does *not* include the traffic that is switched locally by the BRs between the ARs that connect to them in the PoP; in other words it only shows backbone traffic. With this particular matrix the backbone traffic was 75% of the total traffic handled by the core network. Fig. 4.8 shows that the matrix has a couple of peaks where a lot of traffic is sourced by LA and NY PoPs relative to the other PoPs. It also shows that SEA and DEN source very little traffic. This can be attributed to the fact that this traffic matrix was based on the population of the ARs in each PoP, and it is easy to note that NY and LA have high urban and suburban populations.

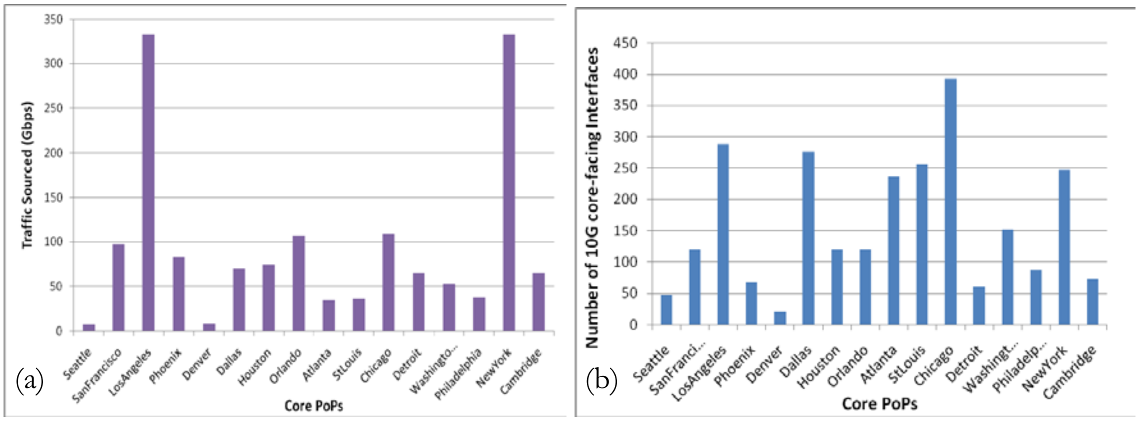
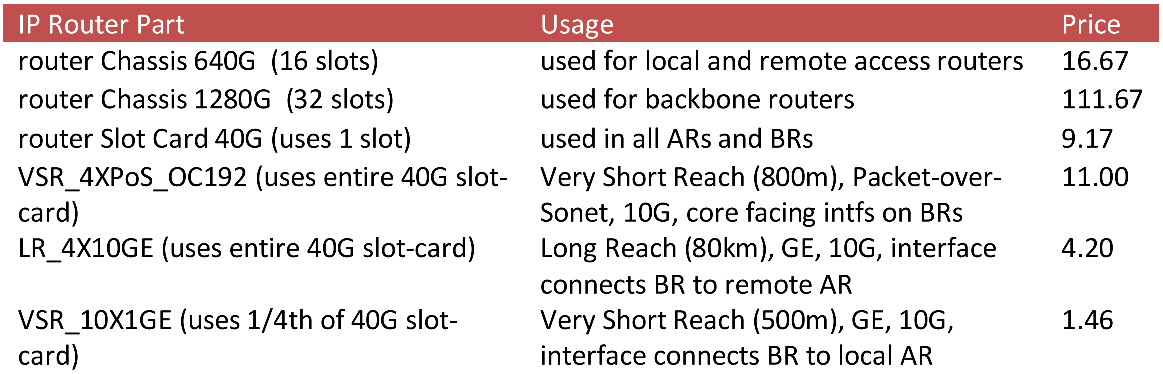


Figure 4.8: (a) Traffic Sourced by each PoP (b) Core facing interfaces per PoP

However, when we compare the traffic sourced (originated) by a PoP to the number of core-facing interfaces in the PoP (Fig. 4.8b), we find a discrepancy in the distributions. While NY and LA have substantial number of interfaces (over 250 each), a significant number of other PoPs have as much or more interfaces (eg. DAL, STL, CHI and ATL). We attribute this discrepancy to the transit traffic being handled at these PoPs, and we will return to this observation in the next section.

Capex Analysis: We need a price model for the network elements in the IP and WDM networks. Normally these industry-prices are unavailable to anyone not actually involved in the procurement of network equipment. Such prices are confidential between carriers and vendors, and can change from customer-to-customer depending on vendor's 'discounts' based on quantity of equipment sold. And so we find that when similar Capex studies are published, the numbers shown are either incomplete [84], or poorly defined [79], or not defined at all [77, 83]. However, we believe that [82] is an excellent reference for a highly-detailed and comprehensive price model. We reproduce the numbers from [82] that are relevant to our study in Table 4.5. The usage column details the part and shows how we use it in the model developed in the previous section. For example, the BRs have a switching capacity of 1.28Tbps and connect to the WDM transponders (for transport to other BRs) using Very-Short-Reach 10G interfaces with PoS framing. The prices listed are all relative to the price of a WDM transponder (listed with a price 1.0). Thus if the transponder costs $1000, then a 1.28Tbps router chassis cost $111,670.



IP Router Part	Usage	Price
router Chassis 640G (16 slots)	used for local and remote access routers	16.67
router Chassis 1280G (32 slots)	used for backbone routers	111.67
router Slot Card 40G (uses 1 slot)	used in all ARs and BRs	9.17
VSR_4XPoS_OC192 (uses entire 40G slot-card)	Very Short Reach (800m), Packet-over-Sonet, 10G, core facing intfs on BRs	11.00
LR_4X10GE (uses entire 40G slot-card)	Long Reach (80km), GE, 10G, interface connects BR to remote AR	4.20
VSR_10X1GE (uses 1/4th of 40G slot-card)	Very Short Reach (500m), GE, 10G, interface connects BR to local AR	1.46

WDM System Part	Usage	Price
LH_10G_TPDR	Long Haul (750km), 10G Transponder (1 per wavelength channel)	1.00
LH_40CH_WDM	40ch WDM line terminal (AWG + pre/post amplifier), bi-directional	4.17
OLA	Optical Line Amplifier (used every 80km), bidirectional, 40ch system	1.92
DCF	Dispersion Comensating Fiber (used every 80km), price is given per km, 750km reach, bi-directional	0.0072
DGE	Dynamic Gain Equalizer (used every 4th OLA), 40ch system, bidirectional	2.17

Table 4.5: IP and WDM network parts Price Model

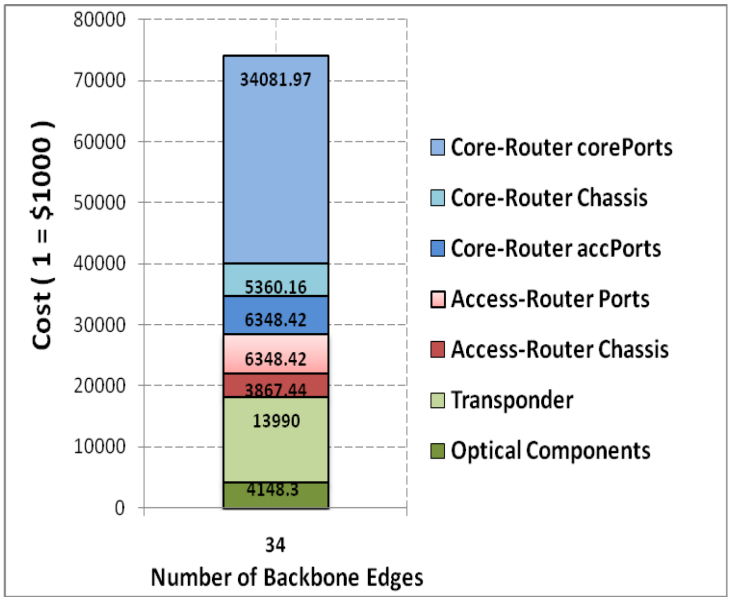


Figure 4.9: Capex Analysis IP over WDM reference design

By applying the price model to the results of our modeling we achieve Capex numbers for the reference IP over WDM design. The results are shown in Fig. 4.9. The WDM part of the Capex (routing 1268 wave demands) totals $18.138 million (assuming a price of 1.0 in Table 4.5 = $1000); 77% of the WDM network cost is attributed to the WDM transponders, and the rest to all other optical components in the lower half of

Table 4.5. The AR related costs are shown for the AR-chassis and the access-ports that connect the AR to the BR. Since we use the same kind of interface (10GE) on both AR and BR, to connect them to each other, the cost of the BR's access-ports is the same. However the BR's core-facing ports total nearly half (46%) of the cost of the entire network (~$34 million out of $74 million)! In retrospect this is not surprising. From Table 4.4, we see that the number of core-router core-facing ports (2564) is greater than the access ports (940+836=1776). And the relative cost of the core-ports (11.00 in Table 4.5) is much higher than the access ports (4.2 and 1.46)[†].

Overall our numbers appear representative – in [84], a 12 PoP backbone with 17 edges, with much-lighter traffic-load (max inter-PoP load was only 8 Gbps – the paper is from 2003) was found to cost $33 million. Our 16 PoP, 34 edge network, dimensioned for 2 Tbps load, and based on a price-model from 2008 [82], costs $74 million. We believe the roughly 2X difference in cost seems reasonable, as it comes from a larger network that also handles a higher traffic-demand.

4.1.3 Effect of Transit Traffic and Static Optical Bypass

We mentioned in the previous section that the number of core-ports in some PoPs is high (Fig. 4.8) for reasons *other* than the traffic that the PoP originates (sources). This discrepancy can be attributed to transit traffic processed by the BRs at each PoP.

Table 4.6 clearly illustrates the *distribution* of traffic in each PoP, as a result of the given traffic matrix and our design methodology. The 1st data-column shows the locally switched traffic; i.e the AR-to-AR traffic that is switched by the BRs in the same PoP, which in aggregate forms nearly 25% of this traffic-matrix (489.49/2000 Gbps). The next two columns specify: incoming traffic from other PoPs demultiplexed (switched) to the ARs by the BRs; and outgoing traffic from the ARs aggregated and switched by the BRs in the PoP. Note that the two columns *total* to be the same value (as they should) and either of them represents the other 75% of the traffic matrix.

[†] The port-cost difference come from multiple reasons including costlier optics, greater buffering, possible use of channelized interfaces, and the use PoS framing in the core ports, versus cheaper GE framing in the access ports. PoS offers OAM capabilities that routers use to quickly detect failures. GE currently lacks such capabilities although standardization is underway to add OAM capabilities in packet networks (by using BFD etc.). We currently do not have a price model for such interfaces.

city-PoP	AR-to-AR	De-mux in	Mux -out	Transit (tr)	Link_fail_tr	node_fail_tr
Seattle	0.10	7.40	10.29	0.00	49.49	0.00
SanFrancisco	40.81	97.23	108.93	4.70	43.09	104.41
LosAngeles	46.72	332.63	321.62	2.72	61.09	21.01
Phoenix	20.74	83.05	83.60	0.00	0.00	0.00
Denver	0.08	7.60	7.26	0.00	19.50	0.00
Dallas	8.35	70.17	69.56	200.03	376.71	345.34
Houston	14.71	74.09	75.35	27.21	208.75	205.87
Orlando	43.62	106.70	115.31	0.92	136.18	113.99
Atlanta	0.67	34.68	37.13	223.55	386.45	356.12
StLouis	1.06	35.84	34.48	162.32	315.92	327.98
Chicago	13.76	108.95	86.18	290.22	461.24	348.02
Detroit	14.34	64.43	69.66	0.00	20.92	0.00
WashingtonDC	1.23	52.64	56.14	126.77	174.67	196.54
Philadelphia	17.23	37.59	34.88	8.55	129.91	103.85
NewYork	243.61	332.78	336.42	3.76	44.03	36.82
Cambridge	22.48	64.72	63.68	0.00	79.06	0.00
	489.49	**1510.51**	**1510.51**			

Table 4.6: Traffic Distribution per city PoP (all traffic numbers are in Gbps)

But the BRs also deal with transit traffic. The 4th data column shows transit traffic at each PoP that arises from SPF routing of the *original* traffic-matrix. Note that some PoPs like Denver and Phoenix do not see *any* transit-traffic, which at first appears strange, given that the cities are somewhat centrally located between PoPs on the west-coast and the mid-west. But this is simply a manifestation of a) the IP topology we have chosen; and b) the use of hop-counts as our SPF metric. Consider this: since we chose not to perform any optimization in our design methodology[†], the edges in the IP topology all have the same-cost; reducing the shortest path calculation to the shortest-hop-count. We also find that from our chosen topology (Fig. 4.1c), the west-coast PoPs (LA, SF and Seattle) are well-connected to the Midwest PoPs (Chicago, St. Louis, Dallas) in *single-hops*. Thus the SPF algorithm always prefers these single-hops over the 2-hop paths transiting through Denver and Phoenix. The choice to ignore load-balancing, traffic-engineering or any other form of path-routing optimization, does *not* change our results and conclusions as we make the same assumptions across all designs.

[†] See design methodology step 3a in Sec. 4.1.1.

According to the design methodology, we also calculate the transit traffic at each PoP that arises from the link and node failure scenarios (we only show the worst case values in the 5th and 6th data columns). Note that even though there may not be any transit traffic in some PoPs from routing the original traffic-demand, under failure scenarios this is no longer true. For example, the Orlando PoP sees transit traffic grow from 0.92 Gbps in normal operating conditions, to 136.18 Gbps under link-failure scenarios.

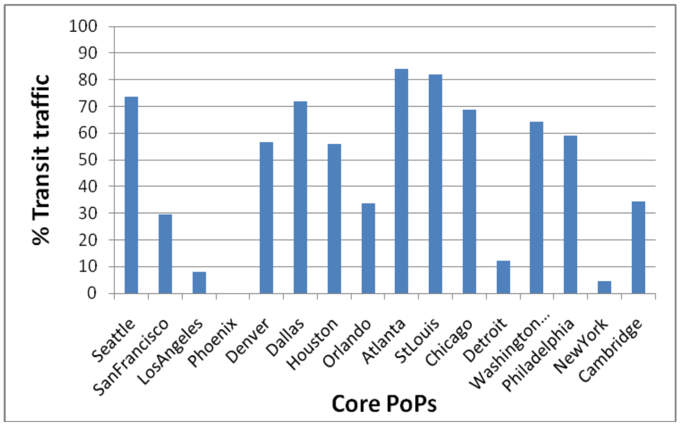


Figure 4.10: Transit traffic as a % of all traffic handled by BRs in city PoPs

In short, the *key insight* here is that a large percentage of BR switching capacity, and correspondingly core-facing interfaces, have to be *dimensioned* to handle transit traffic. Therefore if there is a way to reduce transit traffic in BRs, then we could reduce the number of core-facing ports in the BRs and thus significantly reduce Capex.

Optical Bypass: The industry has proposed the use of optical bypass (sometimes called express-links) to achieve reduction in transit traffic. In optical bypass, the transit traffic remains in the optical-layer (WDM) when passing through a city PoP instead of reaching the IP layer (i.e. the traffic is not routed by the IP router). Two ways to achieve optical bypass are shown in Fig. 4.11. One involves the use of manual-patch-cords between back-to-back transponders in which case the light gets converted to electrical domain in the line-side of the transponder, gets reconverted to optical in the client-side. The client sides are patched together†. Another method involves the use of all-optical

† A small optimization (that we do not consider) involves removing the client-interfaces and patching together the line-sides of the transponders.

ROADMs to *optically* patch (switch) the transit traffic, without requiring the transponders for the transit traffic. This technique requires the use of longer-reach (more than 750 km, more expensive) optical WDM systems as the signal stays optical for a longer distance. Nevertheless, it is vital to note that even though an optical switch is being used, it is set one time to create the bypass and then never changed – i.e. it is a *static* optical bypass mechanism.

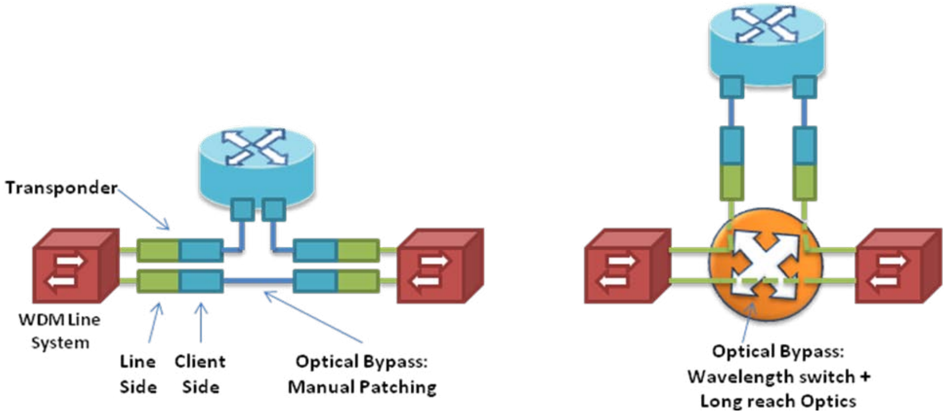

Transponder
WDM Line System
Line Side
Client Side
Optical Bypass: Manual Patching
Optical Bypass: Wavelength switch + Long reach Optics

Figure 4.11: Optical Bypass Techniques

Modeling and Capex-Analysis: Irrespective of the method used for bypass, the net *effect* is the same – an increase in the number of *edges* in the backbone IP topology. With reference to Fig. 4.6, we show in Fig. 4.12 that a new link is introduced in the IP topology due the addition of a wavelength-bypass channel in the transport network.

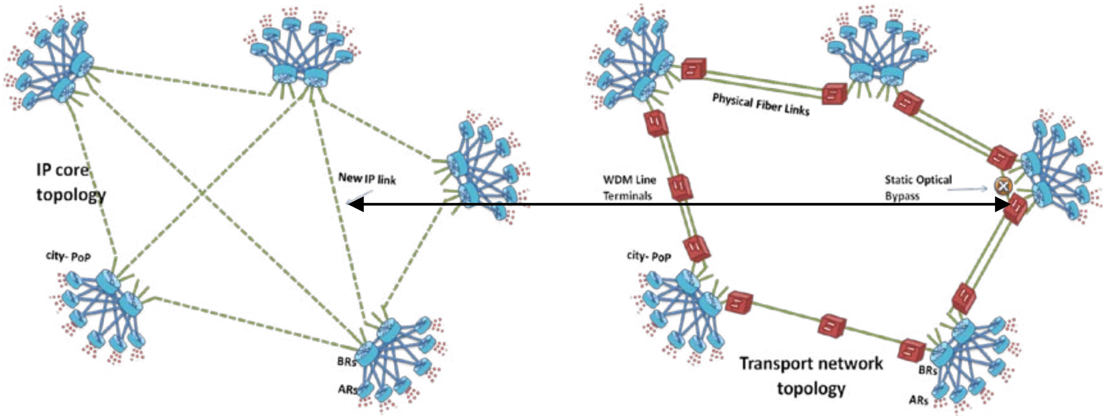


Figure 4.12: Optical Bypass increases 'mesh-iness' of the IP topology

We model this effect in our IP over WDM design methodology by simply incrementing the number of edges in the IP topology from the initial 34 edges (Fig. 4.1c). We iterate over the design process to incrementally add edges (from bypass) from which we benefit the most – ie. the new-edges that reduce the transit traffic the most (over all failure and non-failure scenarios) are chosen first.

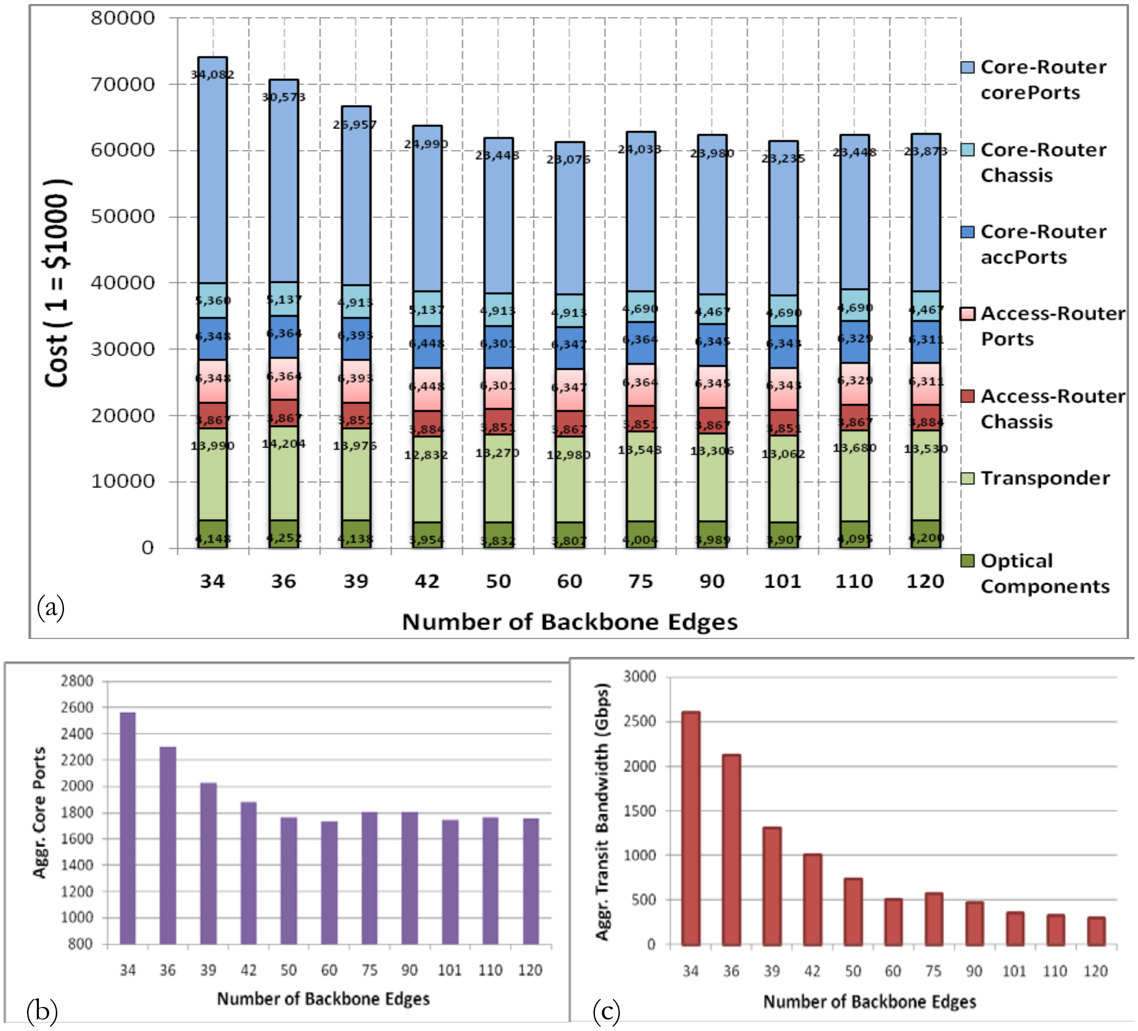


Figure 4.13: Effect of Optical Bypass on (a) Capex (b) CorePorts (c) Transit Bandwidth

Fig. 4.13a shows Capex results for corresponding increase in the number of edges in the IP topology. We find that as we increase the number of edges, there is a reduction in

Capex; but after the initial benefit of reducing large transit cases, we run into diminishing returns. For this traffic matrix, beyond 50 edges, we do not benefit anymore in terms of Capex even if we go up to a fully-meshed IP PoP topology (16 PoPs fully meshed → n.(n-1)/2 = 120 edges for n=16). We take a closer look into this result. From Fig. 4.13a, note that every other category, save for the BR's core-facing ports, does not change appreciably. The major savings, as expected, are in the core-port costs. This cost reduction come from the reduction in the total number of core-ports, which shows the same leveling off beyond the 50-edge case (Fig. 4.13b).

The main reason for this is that the aggregate-transit traffic bandwidth can be reduced significantly, but it cannot be eliminated (Fig. 4.13c). This is because in our design methodology we consider failure cases – consider the fully meshed case[†] (120 edges) – without any failures, all BRs are one hop away from all other BRs, and so there is no transit traffic. But when failures happen, at a minimum, traffic has to be redirected over a 2-hop path, thereby creating transit traffic. Also we show in later sections, that bypass-decisions made for a certain traffic matrix (TM), no longer work as well if the TM varies. Since the optical network is static, it cannot change to varying traffic needs. And so the IP network has to *plan* for such change; thereby potentially reducing savings.

The takeaway point is that static optical bypass can incrementally reduce Capex by ~10-15%, but has limitations in dealing with failures and changing TMs due to its static nature. Thus in the next section we discuss *dynamic* transport network design.

4.2 pac.c Network Design

In this section we present a converged network based on common control over packet-switching and dynamic circuit-switching (DCS). We call such a design "IP-and-DCS" to distinguish it from IP-over-WDM. We first discuss the three main elements of IP-and-DCS design and then explain the benefits of our design-choices in comparison to IP-over-WDM. We then detail the design methodology and show results for Capex and Opex.

[†] As an aside, the fully meshed topology actually results in lesser number of *links* (interfaces) than the original 34 edge case. From Fig. 4.13, 120 edges correspond to ~1800 interfaces, while 34 edges gave ~2600 interfaces. Lesser interfaces result in fewer routing-adjacencies, thereby (counter-intuitively) reducing the load on the routing protocol. Our findings are consistent with [84].

4.2.1 Design Elements

IP-and-DCS design has three main elements:

1. It *replaces* Backbone Routers (BRs) in PoPs, with switches that have both packet switching and circuit switching capabilities in nearly equal measure. We adopt a term used by industry to describe such switches: Packet-Optical Switches† [90].
2. Our design features a *full-mesh topology* between PoPs; created by circuits between the optical-side of the Backbone Packet-Optical Switches in each PoP.
3. And critically, it uses our *SDN inspired unified control-architecture* to commonly control both packet switches and circuit switches in all PoPs.

We discuss each of these in more detail below.

Backbone Packet-Optical Switches: The replacement of backbone routers in PoPs with packet-optical switches is depicted in Fig. 4.14.

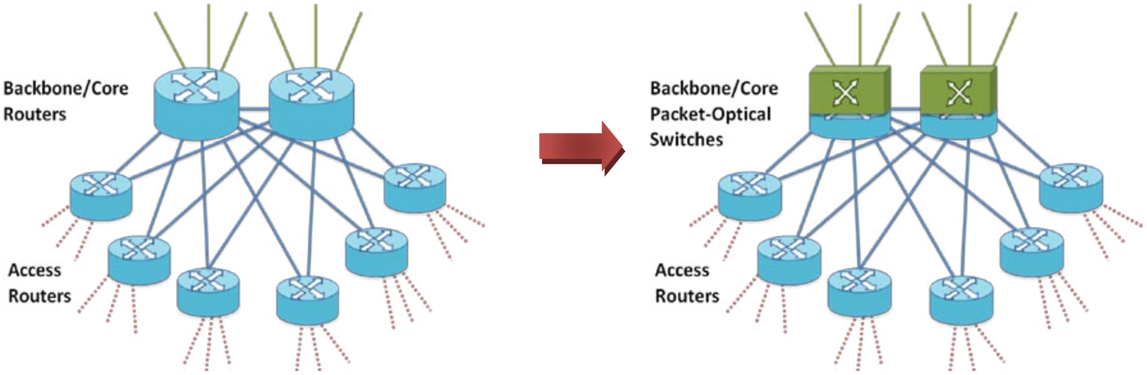


Figure 4.14: Replacing Backbone Routers with Packet-Optical Switches

These Backbone Packet-Optical Systems (BPOS) have the ability to switch-packets and circuits in equal measure – for example, we envision a switch with 1.28 Tbps of switching capacity; half of which is used to switch packets based on MPLS labels; and the other half switches time-slots based on OTN technology [91]. And while such packet-optical switches do not exist commercially at the time of this writing, several companies are indeed working on building such switches [85-88], albeit in a different context [89, 90] (which we shall discuss in the Related Works section in Ch. 6).

† Packet-optical should not be confused with optical-packet-switching. The latter switches packets in the optical domain; in our case packets are switched electronically. Also, digital-electronics is used to switch circuits based on *time-slots* in the 'optical' part of the packet-optical switch. Industry terms such switches as 'optical' even though they possess electronic circuit-switching fabrics. In other words, the terms 'optical' and 'circuit' are used interchangeably in the text. Such systems *may* also include actual optical-fabrics (called *photonic* by the industry) to switch *wavelengths* optically – however we do not consider wavelength switches in this design.

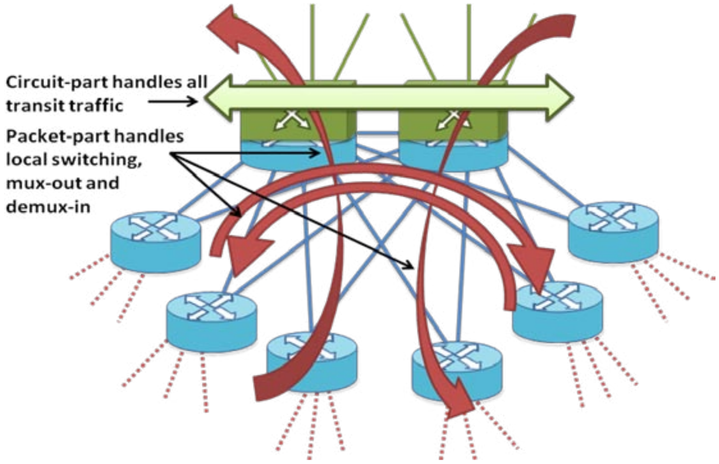

Circuit-part handles all transit traffic
Packet-part handles local switching, mux-out and demux-in

Figure 4.15: Backbone Packet-Optical Switches (BPOS) Usage

The packet-part of the BPOS switches packets between the ARs; multiplexes traffic out to other PoPs; and de-multiplexes incoming traffic from other PoPs (Fig. 4.15) – all functions previously performed by Backbone Routers. However, in our design, we mostly limit the operation of this switch to *tagged* packets, such as those packets already tagged with MPLS labels (by the ARs). We explain the reasoning behind this decision in the next section, after we have described the other elements of our design.

Full Mesh Topology: The basic idea is that we wish to keep all transit-traffic circuit-switched. Accordingly we create an inter-PoP topology that is fully meshed. All PoPs are connected to all other-PoPs directly using dynamic variable-bandwidth circuits.

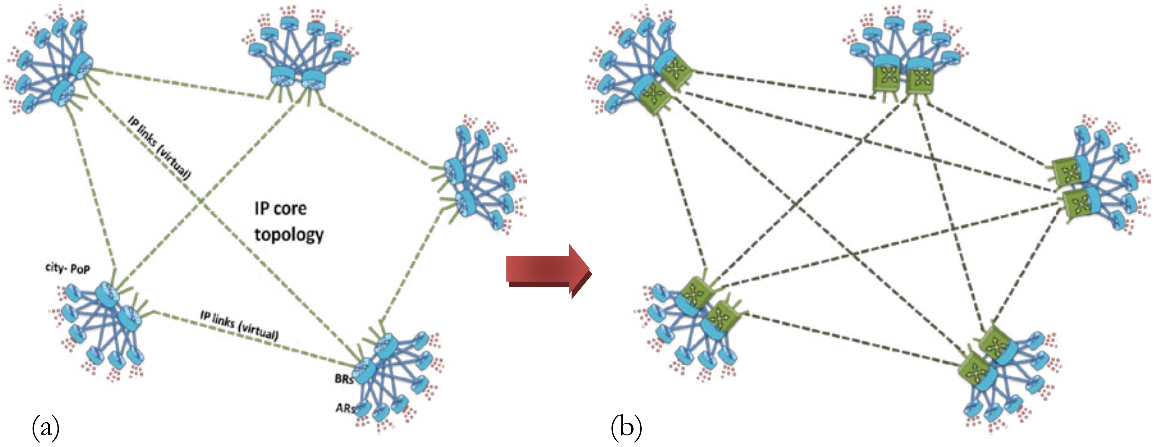


Figure 4.16: Change to Full-Mesh Network Topology

Fig. 4.16a shows the reference IP over WDM design with an IP core topology linking BRs in different PoPs with edges (parallel IP links). In our design (Fig. 4.16b), the BRs have been replaced with BPOSs, and circuits are used to connect all the PoPs, resulting in an IP topology that is fully-meshed. Packets are not allowed to transit through an intermediate PoP by touching the packet-part of the Backbone Packet-Optical Switches (BPOS). Instead they stay in the circuit-layer (see Fig. 4.15), which essentially means that instead of a packet-switched backbone topology, the backbone is now circuit-switched, and it is necessarily fully-meshed. The ARs in each PoP are dual-homed to the BPOSs (as before). However each AR is essentially a single-hop away from every other AR in the entire network; i.e. the ARs form a fully-meshed IP topology. Let us see how – in Fig. 4.17, PoPs A and B, and PoPs A and C, are connected by variable-bandwidth circuits (circuit between PoPs B and C is not shown).

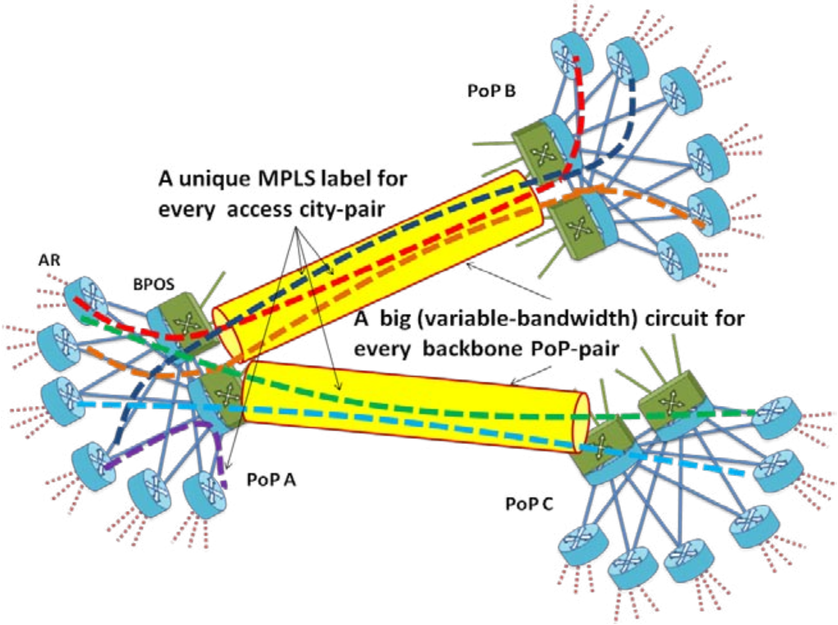

PoP B
A unique MPLS label for every access city-pair
AR
BPOS
A big (variable-bandwidth) circuit for every backbone PoP-pair
PoP A
PoP C

Figure 4.17: IP-and-DCS Network Design

We assign unique MPLS labels bi-directionally to every AR pair. Assigning MPLS labels can make the processing required in the packet-part of the Packet-Optical switches simpler. For example a source AR pushes the corresponding label onto all packets destined for another AR (within the same PoP or in a different PoP), and forwards the packets to the BPOS. The packet part of the BPOS examines the label and forwards the packet to another interface (for local switching) or to the appropriate circuit to the destination PoP. The destination-AR pops off the label and forwards the packet on. It's simpler because the number of rules required in the flow-table for packet-forwarding (in the BPOS) is then on the order of the number of ARs in the network (few hundreds). On the other hand if the rules are on the basis of matching IP addresses then they could number into hundreds-of-thousands. We will explain this in more detail in the next section, after we discuss our control strategy.

SDN based Unified Control Plane: Our design requires the use of a simple, flexible, multi-layer control plane. We use the SDN based common-control plane described in the previous chapters; where all switches including the ARs and the BPOSs support a generic packet-switch and circuit-switch data-abstraction manipulated by a common switch-API like OpenFlow; and a (distributed) Controller creates a common-map abstraction so that network control functions can be implemented from a single centralized application viewpoint.

The requirement for SDN based unified control comes from two facts. First, our core network is as much packet-switched as it is dynamically circuit-switched. Dynamic circuit-switching is used to vary circuit-bandwidths between PoPs for a variety of reasons including recovery, varying-traffic loads, and guarantees. It requires close interaction with packet-traffic and application/service needs, all of which can be provided in simpler and more extensible ways with our unified control architecture (as shown in Ch. 3).

But just as importantly, the SDN based common-control plane *enables the use of the full-mesh topology*, which is an important element of our design. Consider the following: We are not the first ones to suggest the benefits of an IP topology that is fully meshed. In fact in the 90s, several ISPs created IP core-networks that were fully-meshed, where the router-to-router connections were realized using ATM virtual-circuits (PVCs). However such construction suffers from a significant drawback, which contributed to the failure of

IP-over-ATM [92]. The problem is frequently referred to as the O(N^2) issue – when N routers have a fully meshed topology, the distributed link-state routing protocol creates N routing-adjacencies for every router. In this scenario,

- When a link goes down, the routers on both ends of the link inform all their adjacent-routers (~N, i.e. every other router in the network). In turn each of these routers tell all their neighbors (again N), at which point the flooding stops. But it results in O(N^2) messages, which can cause a significant load on router CPUs.
- If the load (in addition to other work the CPU is doing) causes a router to crash, or the router crashes for some other reason, the situation is worse – it generates O(N^3) messages. Such a cascading situation can, in the worst case, crash an entire network.

But note that the O(N^2) issue is *not* due to drawbacks in ATM itself[†]; instead it is an artifact of the nature of distributed link-state routing protocols. With the adoption of SDN, our control-architecture eliminates the use of distributed routing protocols within a Controllers domain. In SDN, when nodes or links fail, the switches inform the Controller of the failure (at worst ~ O(N)). The Controller either re-computes flow-paths or has pre-computed backup paths for the failure. It can download new flow-entries into the affected switches (maybe all switches ~ O(N)) or it could have pre-designated fail-over paths in the switches. In any case the complexity remains O(N). Note that pre-computed backup paths are also possible in today's (non-SDN) networks; nevertheless the N^2 issue remains, simply due to the use of distributed routing protocols. And so the adoption of SDN based control eliminates the O(N^2) issue, and enables the full-mesh IP topology.

4.2.2 Design Benefits

Now that we have covered the three main elements of our design: Backbone Packet-Optical Switches; a fully-meshed IP topology; and the use of an SDN based common control plane; we are in a position to describe the rationale behind the design choices by highlighting their benefits.

[†] ATM had other issues including high-costs; a desire to be an end-to-end network technology; inefficient cell size etc. [93]

Simpler Data-Plane, Simpler Routing: Consider the process of routing in regular IP networks. The first step of the process of getting IP packets to their ultimate destination requires figuring out which router to reach *within* the IP network for the incoming-packet's destination IP address. Such a destination-router is sometimes called the BGP next-hop for that destination address within the network; BGP next-hops for IP prefixes are learnt from I-BGP advertisements. All E-BGP speaking routers in the network have I-BGP connections with each other as well as internal non-E-BGP speakers (Fig. 4.18); this apparently does not scale well in larger networks; so instead, routers maintain an I-BGP session with a Route-Reflector (not shown in Fig. 4.18).

In Fig. 4.18, assume that within the AS shown, for a particular IP-prefix, it has been determined that the BGP-next hop is R4. Then all routers that have incoming packets destined for that prefix need to figure out the next-hop in the shortest path to the BGP next-hop. The IGP (like OSPF or IS-IS) helps determine the topology of the networks, from which shortest paths are calculated and next-hops and outgoing-interfaces are determined. For example, R5 determines that the next hop is R6 to reach the BGP next-hop (R4) for that prefix. All of this is done in *each router* in the network (all ARs and BRs) for *all* the destination IP prefixes in the Internet (nearly half- million).

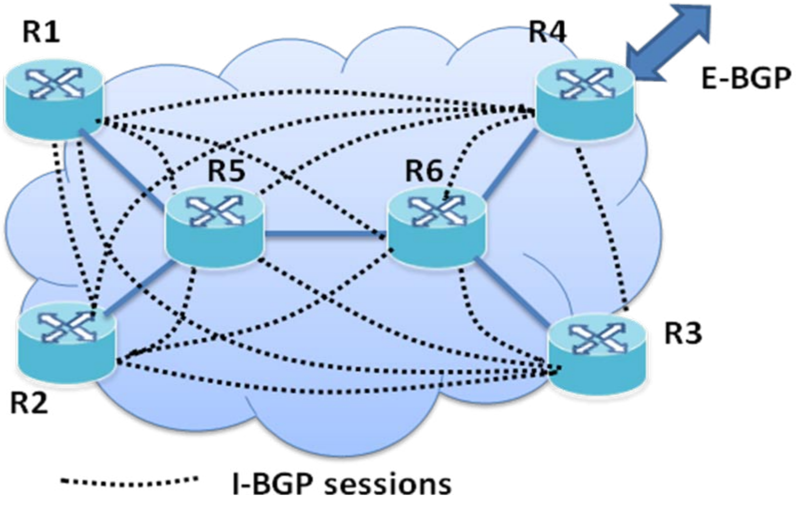
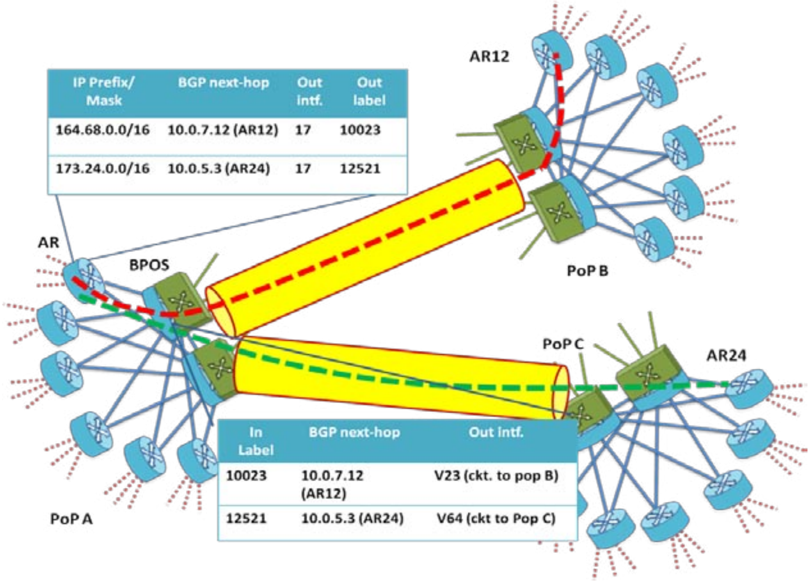


Figure 4.18: Routing in IP networks

In our design, the routers and switches learn about advertized IP prefixes from the Controller, which for this purpose behaves essentially like the Route-Reflector. The Controller discovers the topology; figures out paths from a source-AR to the BGP-next-hop for all IP destination prefixes; and pre-downloads them as matching-rules and actions in the flow-tables of the ARs via the OpenFlow protocol.

The route calculation is trivial as all ARs are one-hop away from all other ARs. The controller adds actions to label the packet for the destination AR (which include peer-routers that peer with other ASes) and forward out of any interface that connects to the Backbone Packet-Optical switches. In the example shown in Fig. 4.19, packets that match two different /16 prefixes are sent to different ARs (AR12 and AR24) by pushing different labels onto them and forwarding to the BPOS (out of interface 17)



IP Prefix/ Mask	BGP next-hop	Out intf.	Out label
164.68.0.0/16	10.0.7.12 (AR12)	17	10023
173.24.0.0/16	10.0.5.3 (AR24)	17	12521

In Label	BGP next-hop	Out intf.
10023	10.0.7.12 (AR12)	V23 (ckt. to pop B)
12521	10.0.5.3 (AR24)	V64 (ckt to Pop C)

Figure 4.19: Routing Tables in IP and DCS design

At the BPOS, the label to destination-AR binding is fixed. For example, packets with label 10023 are always forwarded to a set of virtual-ports (such as V23), which are head-ends of circuits to PoP B (and AR12). These label-to-virtual-port bindings do not change frequently; behind the scenes, if there is a change in network topology, it is the circuits and not the virtual-ports that change. In other words the circuits may be resized or re-routed but they remain pinned to the same set of virtual ports. Further, as new IP-prefixes are learned by the Controller (via E-BGP sessions), the latter downloads the prefix /destination-AR/label information into all ARs without having to do a shortest-path calculation; as all BGP-next hops are one-hop away in this full-mesh topology. In most cases, the rules in the packet-part of the BPOS do not need to be changed.

Therefore the net result is that the data-plane Backbone switches are greatly simplified. The packet-part of the BPOS caches a small number of rules to switch on; which are on the order of the number of ARs in the network (hundreds) instead of the hundreds-of-thousands of rules needed in BRs in a regular IP network.

Note that the gains are not in the *speed*-of-lookup; IP destination-prefix lookup is just as fast today as MPLS-label lookup. Instead the gains are from lowering switch complexity by requiring fewer forwarding rules and actions. This gain is a *direct result of our design choice of using an SDN based control plane in a fully-meshed IP topology*. It in-turn results in highly simplified and inexpensive switch operation in the packet part of the BPOS; leading to the use of cheaper switches which are reflected in our Capex analysis in Sec.4.2.4.

Network Recovery: In Sec. 4.1.3, we saw that transit traffic cannot be eliminated due to recovery considerations when dimensioning an IP network. For example, in Fig. 4.12c, even with 50 edges the transit traffic aggregates to 750 Gbps over the entire network. For a backbone that deals with 1500 Gbps of incoming traffic, an additional 50% of switching-capacity must still be allocated for transit-traffic.

In our design we wish to eliminate *all* transit-traffic – we achieve this with the use of a fully-meshed IP topology. Under normal circumstances, traffic flows directly from one PoP to another in circuits; without having to transit through the packet-part of BPOS in intermediate PoPs. Under failure-conditions, all network-recovery is processed in the circuit layer. Because the circuit-layer is dynamic, it can be re-configured to re-route the packet traffic around the failure in the circuit-domain.

However this decision brings up an important point. Proponents of IP-over-WDM design point out (correctly) that failures in the IP-layer cannot be overcome by recovery methods in the lower (optical/circuit) layer [77]. This subtle point can be better explained with the help of Fig. 4.20.

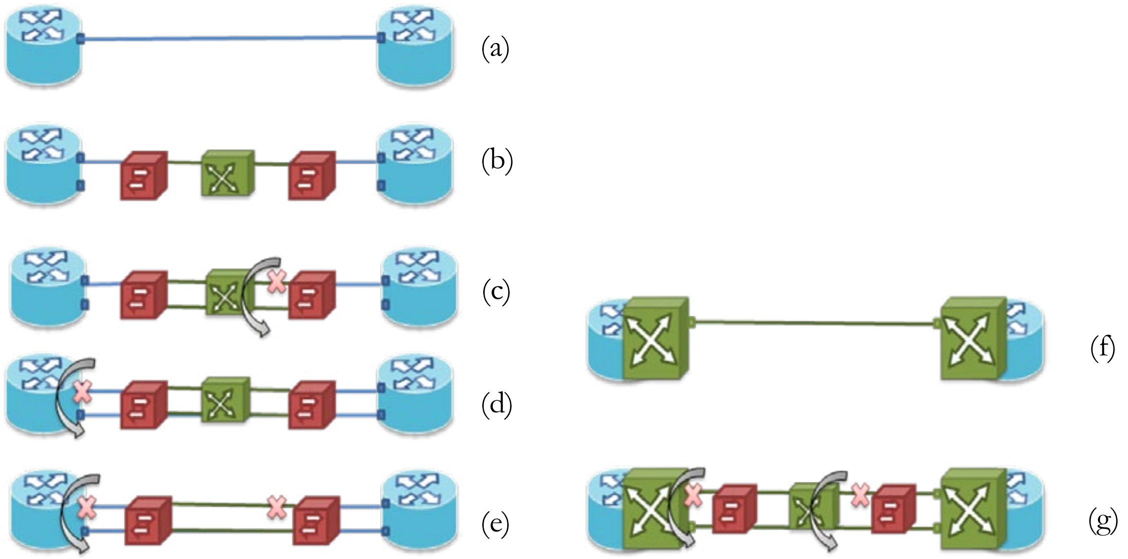


Figure 4.20: Recovery Scenarios

Consider a simple IP link shown in Fig. 4.20a - hereafter referred to as (a). The link is realized by a circuit over the wide-area as shown in (b). Note that in (b), the interfaces on the routers (at both ends of the IP link) are connected to WDM line terminals, and the circuit passes through a circuit-switch in between the line-terminals.

If the optical-link breaks, the circuit-switch can recover from it, as shown in (c), using capacity in the circuit-network. But if the failure is in the IP layer – for example a router interface fails (as in (d)) – then recovery cannot be accomplished in the circuit network. Additional capacity (another IP interface and link) must be provisioned in the IP layer to deal with such failures (as shown in (d)). But provisioned capacity in the IP layer is agnostic to the type of failure. It can recover from both interface failures and fiber cuts (as shown in (e)). And so there is *no need* for the circuit switch or the circuit-layer for recovery (as shown in (e)). Note that (e) is essentially IP-over-WDM, where all recovery is handled by the IP layer, but at the cost of more-expensive Backbone Router core-facing ports.

In our design, BRs are replaced by Packet-Optical switches (as shown in (f)). Importantly, all core-facing ports are now *cheaper* circuit-ports. And crucially, all recovery can be performed in the circuit/optical layer – there is no difference between the loss of an interface or a link – both are provisioned with spare capacity in the circuit network. This fact has significant implications in the Capex analysis in later sections.

Instead of IP-rerouting based recovery methods, or complex MPLS FRR techniques, the circuit network can use established fast and efficient recovery techniques such as shared mesh-restoration [94, 95]. Fig. 4.21 shows how such recovery is accomplished with the technology used in our design. Fig. 4.21a shows the circuits between 3 PoPs. For two of the circuits, it also shows a) the quantization of the circuit into time-slots; b) the usage of the time-slots to satisfy bi-directional demand traffic; and c) the existence of un-used time-slots for recovery and traffic-uncertainties.

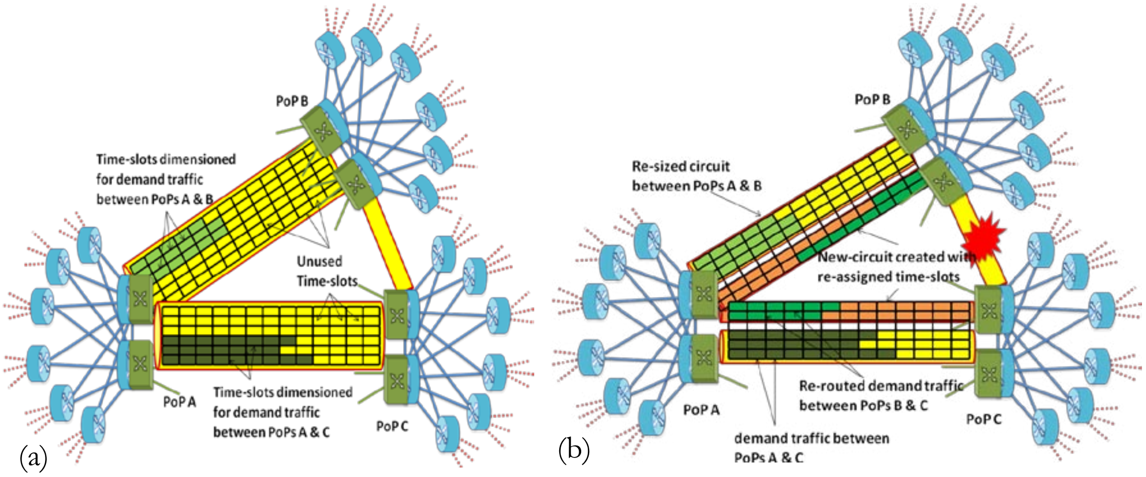

PoP B
Time-slots dimensioned for demand traffic between PoPs A & B
Unused Time-slots
PoP A
Time-slots dimensioned for demand traffic between PoPs A & C
PoP C
(a)
PoP B
Re-sized circuit between PoPs A & B
New-circuit created with re-assigned time-slots
PoP A
Re-routed demand traffic between PoPs B & C
PoP C
demand traffic between PoPs A & C
(b)

Figure 4.21: Recovery in IP-and-DCS Design

It is important to understand that while the circuit is shown quantized into time-slots (which it is); from the point-of-view of the IP network (i.e the ARs) the quantization is not visible – all the ARs see is a single big circuit (of variable bandwidth). Technologies like ODUflex [96] and VCAT [5] make this view possible. Fig. 4.21a shows the normal non-failure case. When the circuit between PoPs B and C fails, the demand traffic between the two is diverted via PoP A. Essentially the spare-time slots in Fig. 4.21a are re-assigned to a newly created circuit between PoPs B and C and circuit-switched at Pop A (Fig. 4.21b).

So far we have shown that our design choices can lead to the use of cheaper circuit ports for recovery. What about performance? IP-over-WDM recovery techniques still require re-convergence of the routing protocol; and while it can be disruptive to traffic elsewhere in the network, it has been shown that such re-convergence can be achieved in sub-second time-scales [97]. Our design uses circuit-switching techniques like shared-mesh restoration which also have been shown to achieve sub-second recovery (~ 250 msec [94, 95]). Backup paths can be pre-computed in both cases and local-recovery can also be performed in 50ms time-scales.

Failure cases *within*-the PoPs are accounted for by dual-homing ARs to backbone switches in both cases. Again in our design it does not require re-convergence of a distributed routing protocol. Recovery is convergence free due to the absence of an IGP; and possibly faster as only the affected traffic is re-routed.

Thus we find that recovery in IP-and-DCS design can potentially lower Capex costs compared to IP-over-WDM design without sacrificing recovery performance, by using circuit-based core-facing ports and circuit-recovery techniques,.

Over-Provisioning for Traffic Variability: In IP-over-WDM, edges are over-provisioned to deal with traffic-uncertainties. Such uncertainties could involve surges in traffic load at certain times or even a shift in the traffic matrix over time.

Consider Fig. 4.22a: when there is a surge in traffic from the ARs in the PoP going out of the BRs, there is no other choice than to have the BRs dimensioned with enough interfaces and switching-capacity to deal with this need. However, if such needs are short-lived (minutes/hours), the extent of over-provisioning in the IP layer *can be reduced* given a dynamic-circuit layer. For example, in Fig. 4.22b, the routers have four interfaces each, two of which are used to carry the average traffic matrix load, and the other two are a result of over-provisioning. Thus, if we assume that the former are nearly always fully utilized, then the latter are used at times of traffic-surge. But if these surges are temporary and at different times one can benefit from a dynamic circuit layer. Such dynamic circuit switching can *redirect the optical bandwidth* to create new links between packet switches to handle surging traffic [98]. We have shown an example of this with our control architecture in the previous chapter, where new packet links were created on-demand between the SF and NY routers. In the simple example of Fig. 4.22c, the same physical interface on any router can be connected to an interface on either of the other routers at different times; thereby saving a core-port per router.

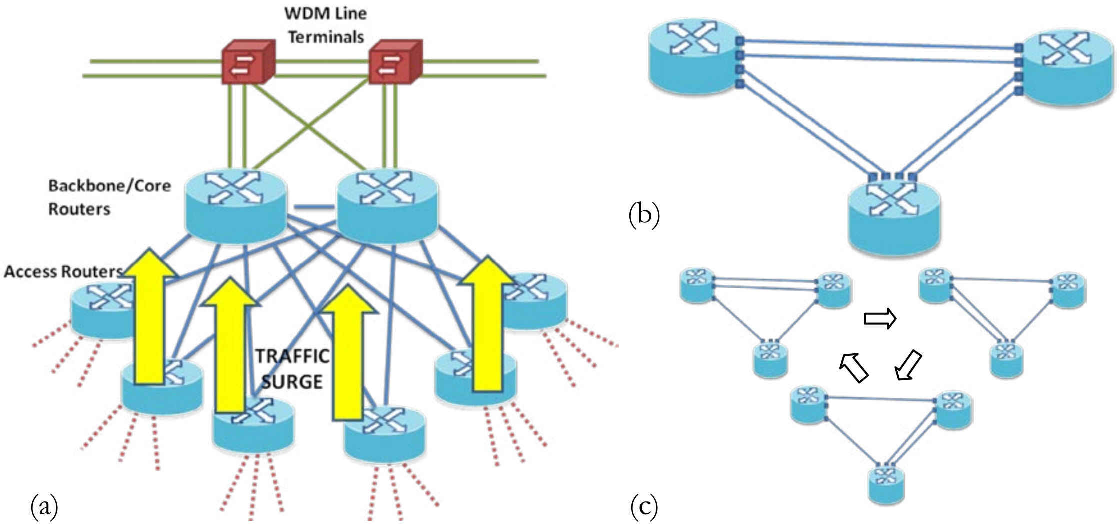


Figure 4.22: (a) Traffic surge (b) Over-provisioning (c) Reduced with DCS

For more long-lived surges (days) or a shift in the traffic matrix, the opportunity to redirect bandwidth is reduced, and the need to overprovision the network increases again. But the advantage of our IP-and-DCS design is that all core-facing ports are circuit-ports which are cheaper. We still over-provision, but we do so with circuit-based core-facing ports. In fact we show (in the next section) that in both designs the amount of over-provisioning is same – the only difference is that one uses IP ports (on BRs) and the other uses cheaper circuit ports (on BPOSs). The quantization of circuit ports into time-slots in the latter case is *not visible* to IP. To packet-traffic, paths to another PoPs just appear as big over-provisioned circuits; in which packet-traffic can grow and diminish seamlessly. Thus under normal circumstances, bandwidth availability is the same in both designs due to the same level of over-provisioning. In fact we can over-provision by a far greater amount and still achieve lower Capex than in the reference case.

And when congestion does happen (despite over-provisioning); a regular IP network has few means for dealing with congestion *reactively* without the use of complex traffic-engineering mechanisms like MPLS auto-bandwidth; but in the IP and DCS case, we can redirect bandwidth between the core-facing circuit ports, from other parts of the mesh. And the same is true if there is long-lasting change in the traffic-matrix.

In other words, over-provisioning and dynamicity makes the design in-sensitive to varying traffic patterns, while still achieving lower costs; with the net result that bandwidth availability in IP-and-DCS design, is at least the same, or in most cases better than the reference IP-over-WDM design.

Delay & Jitter: This performance criterion is a clear-win for the IP-and-DCS design as all AR to AR communication is a single (packet) hop. The circuit network has propagation delay (which cannot be avoided and exists in all networks); but typically removes the switching delay inherent in packet networks. Thus overall latency is reduced, bounded and guaranteed by the circuit-path, while jitter is eliminated.

4.2.3 Design Methodology

The design methodology for IP-and-Dynamic Circuit Switching (DCS) design is very similar to the methodology for IP-over-WDM design outlined in Sec. 4.1.1. But some of the steps have to be modified to account for our design choices of replacing BRs with a fully-meshed DCS network. We overview the steps first and then give details:

1. Network Topologies: The WDM/fiber topology remains the same; but the IP PoP structure changes according to Fig. 4.14; and the IP inter-PoP topology changes to reflect the full-mesh (Fig. 4.16).

2. Traffic-Matrix: We use the same traffic-matrix.
3. IP Edge-Dimensioning: In this step we account for all the traffic that could traverse an *edge* in the IP topology. The difference is that there is a full-mesh of edges. As before, we route the average traffic-demand between PoPs; account for failures; and on the edges.
4. IP PoP-Dimensioning: Once all the edges in the IP topology have been dimensioned, we calculate the number of parallel 10G *links* that make up the edge; and the number of Access Routers and Backbone Packet-Optical Switches required in the PoPs to switch between those links.
5. WDM-Network Dimensioning: Finally each 10G link (actually 10G OTN circuit) in the IP network is translated to a 10G wavelength path in the WDM network. The path is determined by routing the circuit over the WDM (fiber) topology. Once all links have been routed, we calculate the number of WDM line-systems, optical components and WDM transponders required to satisfy the demand.

We give details of each step in the design-methodology:

1. Network-Topologies for the IP network and WDM network:
   a. We use the same IP PoP locations that we used in the reference design. The access-router locations are the same as well. There major differences are :
      i. BRs in the PoPs have been replaced by BPOSs;
      ii. The edge topology from Fig. 4.1 is no longer used. Instead we assume a fully-meshed IP PoP topology. Since there are 16 PoPs, the IP topology has 120 edges.
   b. The WDM topology remains the same as the one used in the reference design (Fig. 4.2).
2. Traffic-Model: We use the same traffic-matrix as the one used in the reference design for fairness in comparison. We also vary the matrix for both designs and scale to 10X in our comparison.

3. IP-Edge Dimensioning: As before we first dimension for the traffic-demand, then we account for recovery, and finally we over-provision for traffic uncertainties.
   a. Dimensioning for demand-traffic is simple. There is no need to run a routing algorithm as every AR is 1-hop away from every other AR. We consider each PoP pair separately, and aggregate the demand from all the ARs in a PoP to ARs in the other PoP in both directions. We pick the larger of aggregated-demand and set it as the bidirectional-demand for the PoP pair. This inter-PoP IP edge is actually realized by a TDM circuit. Thus we calculate the number time-slots needed on the edge to satisfy the bi-directional demand with TDM technology based on OTN [91]. We assume a minimum switching granularity of ODU0 (1.25 Gbps) and the use of ODUflex [96] to treat all time-slots (even on multiple waves/interfaces) as part of the same 'circuit'.
   b. Dimensioning for Recovery: The advantage of having all core-facing ports be circuit switched is the circuit-domain can recover from all core- Shared mesh restoration is basically the use of a backup path for a collection of used paths – the restoration capacity along links is shared between the used paths. In the single failure case, the there is no contention for resources in the backup path. We use this logic in developing a simple shared-mesh-restoration algorithm.

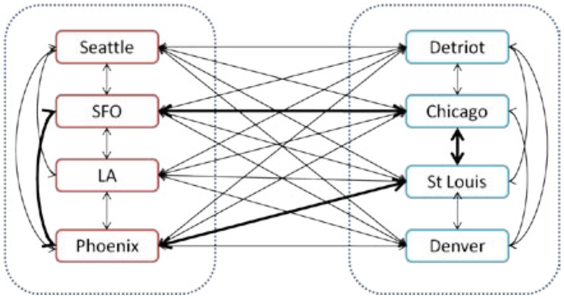


Figure 4.23: Shared Mesh Restoration Example

As in the reference case, we assume only single-failure-scenarios and we also do not optimize the algorithm for any particular criteria. Our algorithm has the following steps:

i. We first separate the PoPs into groups of four based on geographical proximity. Fig 4.23 shows two such groups. Then we consider the demands between each pair of groups, and find the PoPs with the maximum demand traffic between the groups. In Fig. 4.23, the edges between SFO-Chicago and Phoenix-StLoius have the most demand traffic. We backup these edges up with each other, both in the inter-group edges and the intra-group edges. For example, if the SFO-Chicago edge fails, it is backed up by SFO-Phoenix-StLouis-Chicago route.

ii. Next we consider each PoP's links that go out of the group and back those up with one of the two inter-group paths selected above. For example, we pick the highest demand on Seattle's external edges to Detroit, Chicago, StLouis, and Denver and back that up on to Seattle-SFO or Seattle-Phoenix.

iii. Finally we consider the edges internal to each group. For each edge we back up the demand with a path within the group.

iv. It is important to note that the recovery capacity does not add up on an edge for each failure scenario described above. Instead it simply picks the worst case backup capacity required for any one of the scenarios. This comes from our single failure assumption which was also used in the IP over WDM reference design.

c. Finally we dimension for traffic uncertainties by over-provisioning the total traffic and recovery demands by the same 4X over-provisioning factor we used in the reference design.

d. Note that the AR to BPOS edges within the PoP are dimensioned exactly the same way as they were for the AR to BR edges in the reference design.

4. IP-PoP Dimensioning: Now that we have dimensioned each edge in the full-mesh PoP topology, we can figure out the number of 10G links that make up the edge. Further we can calculate the number of Backbone POS' required and the number of ARs per PoP.

a. The number of parallel backbone links per edge is found by dividing the dimensioned time-slots per edge (demand + recovery) by the number of time-slots that fit in a 10G interface. Since we assumed that each time-slot is an ODU0 (1.25 Gbps) and eight of these fit in an ODU2 (10Gbps), we can calculate the number of ODU2 interfaces (or rather OTU2 interfaces/waves†) needed.

b. The access links and access routers are determined by exactly the same procedure as the IP over WDM reference design.

c. Finally the number of backbone packet-optical switches are determined first figuring out the number of OTN switches (640 Gbps) required to satisfy the core-interfaces; and the number of core-packet-switches (640 Gbps) required to satisfy the access interfaces; and picking the greater of the two as the number of packet-optical switches with 1.2Tbps of switching capacity (with half for packet and the other half for OTN).

5. <u>WDM system requirements</u>: The final step of the design methodology is to route the circuits that make-up the full-mesh core topology on the fiber topology (Fig. 4.24).

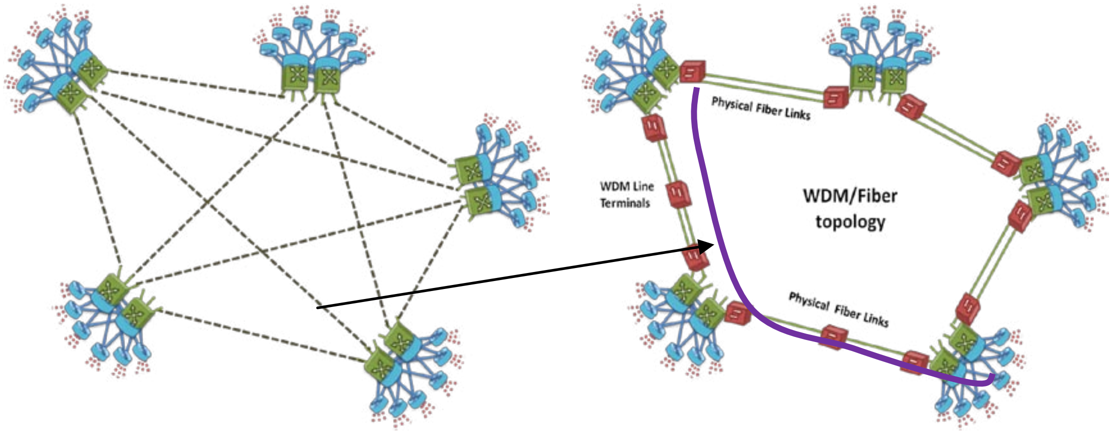


Figure 4.24: Routing Circuits on the Physical Topology

This procedure is exactly the same as the one followed for the reference design, the only difference being that the waves in the reference design supported 10G interfaces on the Backbone Routers, and here they support 10G interfaces on the OTN part of the Backbone Packet-Optical Switches.

† In OTN terminology, ODU is the data unit or payload (ODU2 = 10.037274 Gbps) and OTU is the transmission unit which includes the payload and the overhead (OTU2 = 10.709335 Gbps)

4.2.4 Results and Capex Analysis

The overall number of core-ports is reduced in IP-and-DCS (1480) when compared to the reference design (2564). This has obvious Capex benefits; as the core-facing ports form the largest share of the Capex in an IP network.

It is also worth noting that the *distribution* of core-facing ports across PoPs (Fig. 4.25) has changed, compared to the reference design (Fig. 4.8). Both figures show (on the left side) the same distribution of traffic sourced/originated by each PoP. Essentially this reflects the PoP to PoP traffic-matrix used, which is the same for both designs. However on the right-side, each figure shows the respective core-facing port distribution. What stands out is that in the reference design (Fig. 4.8b), the distribution only lightly resembles the traffic matrix, with PoPs in cities like Chicago and Atlanta using far more ports than needed to satisfy the sourced (or sunk) traffic. This of-course is a manifestation of all the transit traffic handled by these nodes.

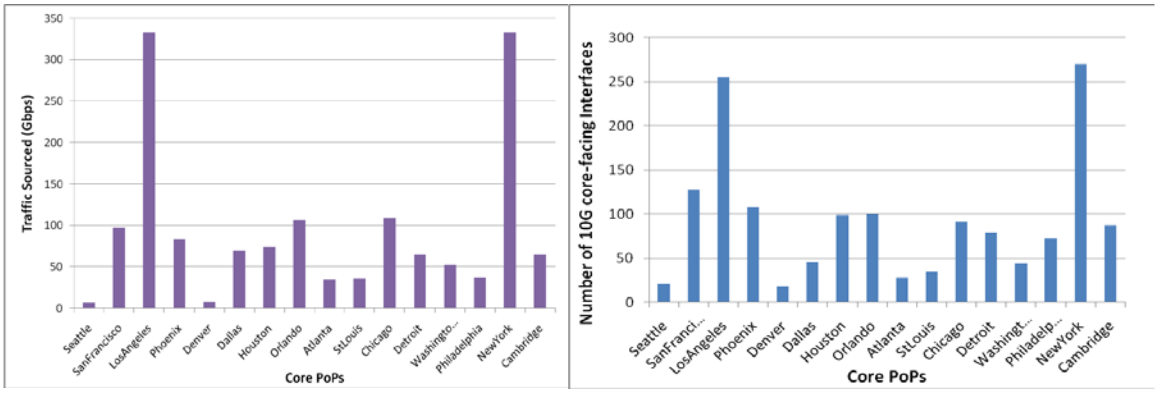


Figure 4.25: (a) Traffic Sourced by each PoP (b) Number of core-facing Interfaces

In contrast, Fig. 4.25b shows a far greater similarity to the traffic-matrix distribution. Note that the largest use of core-ports is in the cities that source the greatest amount of traffic (LA and NY). There are two reasons for the similarity: a) the overall amount of transit traffic has been greatly reduced by the full-mesh circuit topology (and in-turn the full mesh AR-to-AR IP topology); and b) transit traffic in the presence of failure scenarios has been reduced due a combination of: all transit traffic being circuit switched; and the use of shared-mesh-restoration in the circuit network (by definition a traffic-engineered network).

To quantify the Capex benefit, we assign a cost (actually price) model to the component parts of our design. In Table 4.7, we detail the cost model for the IP routers and Backbone Packet-Optical Switches in the PoPs. The costs for the WDM network are the same as in the reference network (Table 4.5). The Access Router costs are the same as those used in the reference design (a subset of the costs shown in Table 4.5). All costs are derived from the extensive cost-modeling done in [82]. The Access Routers have a switching bandwidth of 640Gbps with 16 slots each with a slot-capacity of 40 Gbps. When in locations (cities) remote from the core-city-PoPs, they use long-reach 10GE ports to connect to the core-switches; when local they use very-short-reach 10GE ports (emulated as 10 GE ports).

The core-switches are hybrid Packet-Optical switches. They have a packet-switching part with switching bandwidth of 640 Gbps, and an OTN based TDM switching fabric with another 640 Gbps of switching bandwidth, to give a resultant 1.28Tbps of switching capacity. We deemed the packet-switching part to be simpler than the ARs, with switching being limited to MPLS labels. As such they are similar to the switches being discussed as Carrier-Ethernet or MPLS-TP switches, and so they are cost-modeled in the same way as in [82]. The circuit-switching part is modeled with OTN switch-fabric and 10G (ODU2) interfaces from [82].

Access Router Costs	Usage	Price
router Chassis 640G (16 slots)	used for local and remote access routers	16.67
router Slot Card 40G (uses 1 slot)	used in all ARs	9.17
LR_4X10GE (uses entire 40G slot-card)	Long Reach (80km), GE, 10G, connects remote AR to backbone switch	4.20
VSR_10X1GE (uses 1/4th of 40G slot-card)	Very Short Reach (500m), GE, 10G, connects local AR to backbone switch	1.46

Backbone Packet-Optical Switch Costs	Usage	Price
Packet Part:		
Carrier-Ethernet Switch Chassis 640G (16 slots)	packet part of Backbone Packet-Optical Switch	28.125
Carrier-Ethernet Port Card 4X10GE	uses 1 slot	8.330
10GE_XFP_10km_Reach	to connect to loca ARs	0.050
10GE_XFP_40km_Reach	to connect to remote ARs	0.125
Optical Part:		
OTN Switch Chassis 640G (16 slots)	optical part of Backbone Packet-Optical Switch	13.330
SR-ODU2 (2km reach, 1310nm)	short-reach 10G core-facing interface, connects to WDM transponders	0.670

Table 4.7: IP and DCS network parts Price Model

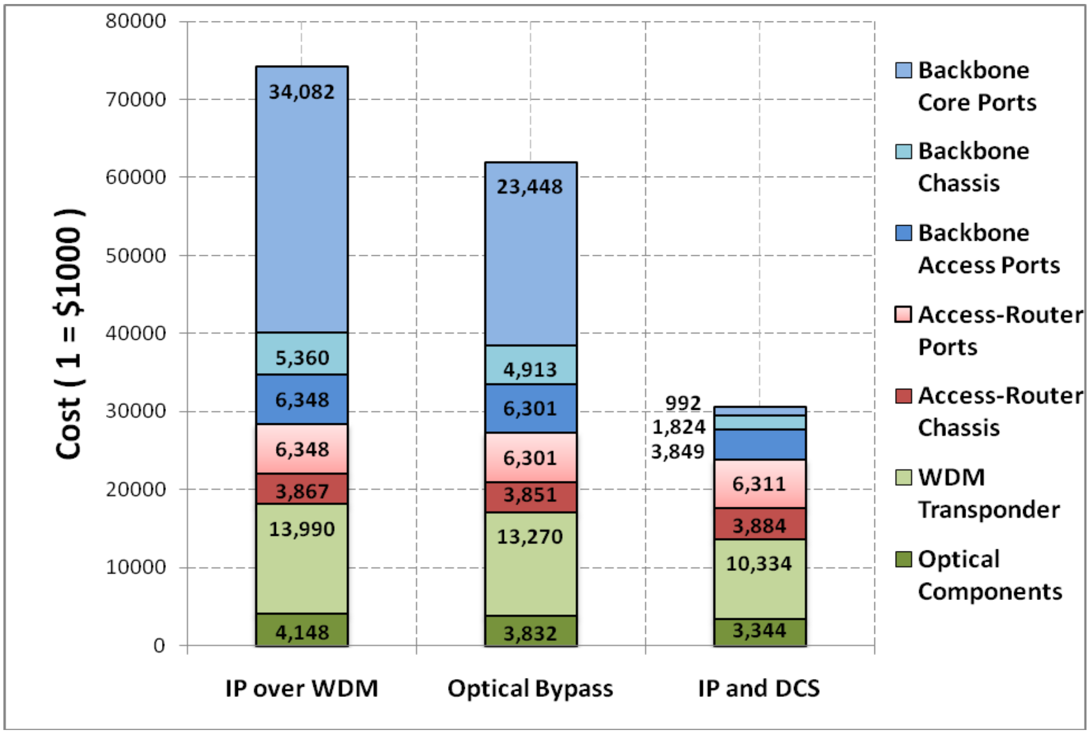


Figure 4.26: Capex Analysis

The overall results of our Capex analysis can be seen in Fig. 4.26. With design choices made in IP-and-DCS design, we achieve 59% in overall Capex savings when compared to the reference IP-over-WDM design. Most of these savings come in the backbone switches, which see an 85% reduction in cost (these include the backbone chassis and access & core-facing ports).

The reduction comes not only from the design choices that lead to requiring *fewer* core-facing-ports, but also from these ports being circuit switched and therefore a lot cheaper. This last fact is especially relevant when over-provisioning the core network for traffic variability. In both the reference design and the IP-and-DCS design, we over-provisioned the network by limiting the utilization to 25% of capacity. But this 4X over-provisioning was achieved with IP core-ports in the reference design (relative cost = 11.0); whereas in IP-and-DCS design we used SR-ODU2 OTN core ports which are much cheaper (relative cost = 0.67).

We also see a 25% reduction in WDM system costs (transponders and optical components). This reduction can be attributed to the design choices of full mesh topology with shared-mesh restoration; that ultimately result in fewer 10G “waves” required from the WDM network. In the reference design 1268 10G waves are routed in the WDM network. In our final design only 740 10G waves are needed leading to fewer WDM systems and corresponding transponders.

It is also worth pointing out that our design achieves 50% in overall Capex savings when compared to the IP-over-WDM design with Static-Optical-Bypass (middle column in Fig. 4.26). This is directly a result of the use of a dynamic circuit switched network as opposed to a static optical network to supplement the IP network. The static bypass network achieves savings by reducing transit-traffic and IP core-ports. But it is still completely a packet-switched IP network with recovery and over-provisioning achieved using IP core-ports and packet-switch-fabric bandwidth. Alternatively, these functions in our design are achieved with a combination of packet-switching and cheaper OTN based circuit switching.

Varying Traffic-Matrices: The benefits of dynamicity in the optical network can be readily seen when we vary the traffic-matrix (TM). Fig. 4.27 shows the results of our Capex analysis for three different TMs (same aggregate traffic-demand of 2 Tbps).

In Fig.4.27a, we show the traffic sourced by each PoP for the three TMs.

- TM1 is the original traffic-matrix (with peaks in NY and LA) with which we have referred to in all the analysis presented thus far.
- TM2 shows a more evened out distribution of traffic with smaller peaks (NY is highest but San Francisco, Orlando, Chicago, St Louis etc. are high too).
- TM3 is less balanced, like TM1, but with peaks in completely different cities (Chicago and Washington DC) compared to TM1.

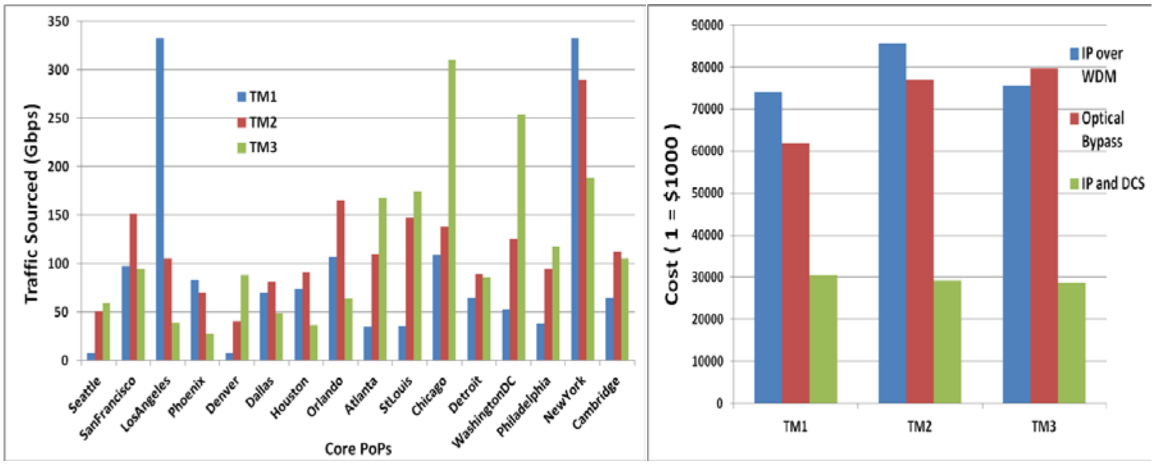


Figure 4.27: (a) Traffic sourced by each PoP for 3 Traffic-Matrices (b)Capex Analysis

Fig. 4.27b shows the overall Capex results for each design with the three traffic matrices. The columns for TM1 are a reproduction of Fig. 4.26 showing 59% and 50% Capex savings of IP-and-DCS design when compared to IP-over-WDM design without and with Static-Optical-Bypass respectively.

The Capex columns for TM2 and TM3 show different trends for the designs. For TM2, the traffic matrix is more evenly distributed. It results in more IP core-ports in each PoPs for both sourced traffic and transit traffic, which in turn, results in higher Capex

($11 million more than for TM1). TM3 which is less-balanced results in reference design costs similar to TM1. But irrespective of the traffic-distribution the IP-and-DCS design yields nearly the same Capex costs for each matrix; reflective of how the design mirrors the matrix by reducing transit traffic and uses cheaper dynamic-circuit-switched core-facing ports. Also with the same bypass-choices used for static-optical-bypass from the TM1 distribution, TM2 shows lesser savings 10%, while TM3 results in a complete erosion of savings from bypass – the static bypass case is actually more expensive than the reference design. This is obviously a result of using the bypass-candidates selected for TM1, in the Capex analysis for TM2 and TM3. Normally we would have chosen different bypass candidates from which we would have gained the largest benefits for TM2 and TM3.

But we did not do so, to highlight a problem with static bypass. With static-bypass the design decision to create bypass is done offline-and-beforehand and then put into place. But if the traffic matrix changes significantly (as in going from TM1 to TMs 2 and 3), the static-bypass decisions cannot be changed – we are stuck with it. And traffic matrices can change significantly as the Internet traffic distribution is no longer population-centric – for example a new data-center in Oregon can significantly increase the traffic sourced/sunk by the Seattle PoP (see Seattle traffic for the 3 TMs in Fig. 4.24a). Even in the short term traffic-matrices have been hard to predict. And so if bypass decisions are made-statically, and the traffic matrix can change, the IP network has to *plan* for such change, thereby reducing the savings from bypass. On the other hand, our IP and dynamic-optical network design is insensitive to the changes in the traffic matrix irrespective of how great the change may be.

Scaling Traffic-Load: Finally in Fig. 4.28 we show the effect on Capex, of scaling TM1 to five times the original aggregate bandwidth demand. Scaling the traffic-matrix is an effective way to plan for increasing traffic-demand. When decisions are made to upgrade a backbone network with new purchases of equipment (Capex), they are often done so with future traffic growth in mind. No one upgrades a network every year.

Thus equipment is put in place such that the network can deal with increasing year-to-year traffic and upgrades that are disruptive to operation are not required for a few years.

So if we plan for 10Tbps traffic instead of the current 2 Tbps, we see that our design affords nearly $200 million in savings. Importantly we find that the Capex costs are *diverging* with increasing traffic-demand. Our design choices lead to a Capex vs. Demand slope of $11million/Tbps, which is significantly lower than the slope for IP-over-WDM with ($23million/Tbps) and without ($29million/Tbps) static optical-bypass.

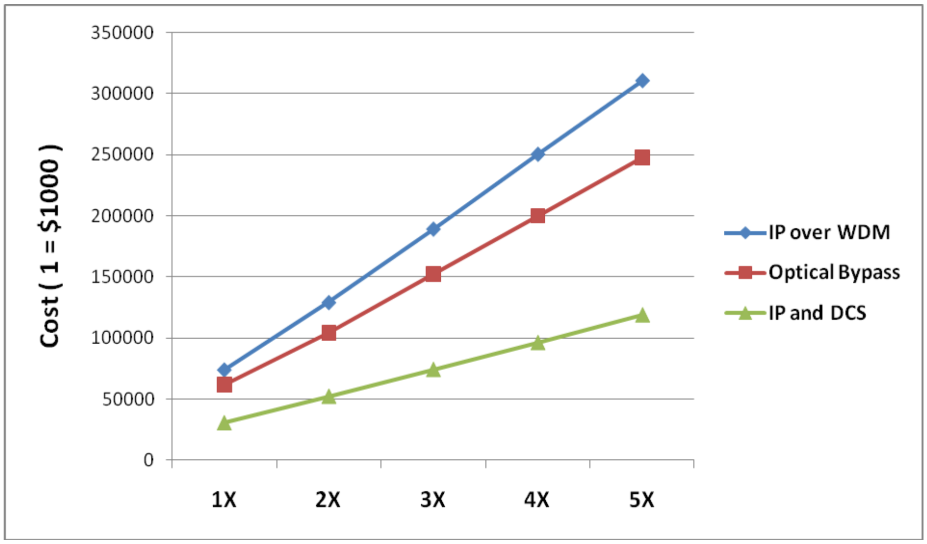


Figure 4.28: Capex Analysis for scaled Traffic Matrix

Control-Plane Costs: The Capex results presented thus far for the IP-and-DCS design do not show the cost of the control-plane. Since our design is based on SDN, the control-plane is de-coupled from the switching hardware (for both packet and circuit switches), and hosted in an external Controller. The Controller itself is distributed over multiple servers in multiple geographic locations. Nevertheless, we assume the Controller costs to be very small compared to the switching-hardware costs for two-reasons – a) commodity server hardware costs are small (< $5k) compared to switch hardware costs ($100k-$1million); and b) the cost for controller software-development is already amortized into the switch-hardware cost in Table 4.7, because our numbers are from [82].

4.2.5 Opex Analysis

Analysis of Operational Expenditures is inherently difficult as it includes a wide variety of factors that can only be truly understood, and accounted for, by the network operators. Such costs include labor costs, equipment rack and building rentals, and power consumption for operation and cooling, among other things. Many of these costs are subjective and may vary widely from provider to provider, depending on network-size, geographic location, labor laws, electricity costs and other factors.

One factor that probably forms a large part of Opex is the *hidden-costs* of time and engineering-labor intensive manual coordination between IP and transport network teams. With our unified control architecture, we can potentially have the greatest impact on reducing labor-costs by running an automated and unified control plane for different switching technologies. Instead of maintaining two teams of operators trained on different modes of operation with different management tools, a single team can accomplish such tasks. Furthermore our architecture eases and speeds-up the introduction of new functionality and services across packet and circuit networks, potentially leading to lower Opex and faster revenue generation. Unfortunately we have no means to gather or gauge such proprietary data and hidden costs.

Nevertheless all service providers stress that Opex accounts for nearly 60-80% of their Total Cost of Operations. And so we attempt a *limited*-Opex analysis by clearly stating our assumptions and focusing on a small subset of all that gets accounted for as Opex. Our goal is to show the potential cost savings that may be realized from our design when compared to the reference IP-over-WDM design. We focus on power consumption for backbone-equipment, labor costs for maintaining such equipment and possible rental costs for hosting the equipment in PoPs.

The backbone equipment quantities shown in Table 4.8 are the ones we obtain from our Capex analysis in the previous sections. Note that quantities are shown for both the original aggregate-demand of 2Tbps, as well as the 5X scaled demand of 10Tbps.

1X	Core Router/Switch	WDM Systems	Transponders
IP over WDM	48	235	13990
Optical Bypass	44	218	13270
IP and DCS	44	184	10334
5X	Core Router/Switch	WDM Systems	Transponders
IP over WDM	180	915	69090
Optical Bypass	148	867	65024
IP and DCS	152	589	43158

Table 4.8: Equipment Quantities for each Design (1X and 5X Traffic Matrices)

The IP-over-WDM network (with and without Optical Bypass) uses 1.28 Tbps Backbone Routers modeled on the Cisco CRS-1 routers [99]). The IP-and-DCS network designed in Sec. 4.2.3 uses Backbone Packet-Optical Switches. As mentioned before, such switches have not been released commercially, but several vendors have declared that they are actively creating such products. We model this switch on one such vendor's product offering (Ciena 5400 [100]). Both designs use generic WDM equipment and WDM transponders. We have left out the Access Routers in this analysis, as they are common to and number the same in both designs. In Table 4.9, we show the power-consumption (in kW), OAMP hours/week, and rack-space required for the equipment†.

Equipment	Power (kW)	OAMP hours/week	Units/Rack
Backbone Router	28.06	24	0.33 (need 3 racks)
WDM System	0.52	1	6
WDM Transponders	0.07	0.25	36
Packet Optical Switch	10	12	1

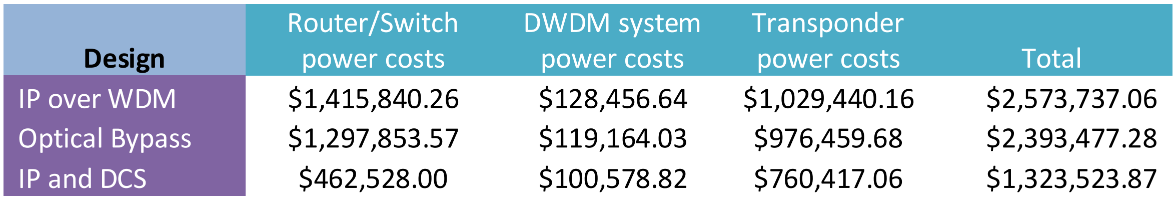
Table 4.9: Opex Parameters

Power-Consumption: The easiest Opex parameter to gauge is power consumption, as equipment specifications clearly state the DC power consumed in fully-loaded chassis. Note however that this does *not* include the power required for cooling which can be significantly higher; but again is a more subjective number depending on cooling type, rack alignment, air-circulation etc.

† All the numbers in Table 4.9 have been obtained either from spec-sheets or from private conversations with industry.

Backbone routers modeled on state-of-the-art CRS-1s' achieve 1.28Tbps switching bandwidth in multi-chassis configurations. This is reflected in the Capex analysis (Table 4.5) as well as the reference from which we derive the cost-matrix [82]. Each 16-slot single-shelf-system houses 640 Gbps of switching capacity and multiple such chassis can be tied together with a fabric-card-chassis to scale the system [99]. Thus the power consumption of a 1.28 Tbps system is the sum of the power consumptions for *two* single shelf systems (9.63kW each) and a fabric-card-system (8.8kW). For the Backbone Packet-Optical switch, we assume 8.5kW for 640Gbps of packet-switching and 1.5kW for 640Gbps of OTN based circuit-switching. The switching bandwidth can be attained in a single-shelf system such as the 5400 [100] †.

Electricity costs in the US can vary widely from state to state (from 7 cents/kW-hr in North Dakota to 26 cents/kW-hr in Hawaii) [101, 102]. For our calculations we use a modest 12 cents/kW-hr. Table 4.10 shows the *annual* power-consumption costs for all three designs for the original traffic matrix (aggregate 2 Tbps traffic).



Design	Router/Switch power costs	DWDM system power costs	Transponder power costs	Total
IP over WDM	$1,415,840.26	$128,456.64	$1,029,440.16	$2,573,737.06
Optical Bypass	$1,297,853.57	$119,164.03	$976,459.68	$2,393,477.28
IP and DCS	$462,528.00	$100,578.82	$760,417.06	$1,323,523.87

Table 4.10: Annual Costs for Power Consumption

It is worth noting from Table 4.10, that our IP-and-DCS design can nearly halve the overall annual power-consumption costs, with most of the savings coming from the replacement of Backbone Routers with Packet-Optical switches. Also worth noting is that the WDM transponder power costs are very high. In IP-over-WDM it is close to the BR power consumption, and in IP-and-DCS it exceeds the backbone switch power consumption. This is simply because of the need for thousands of such transponders in the WDM network (see Table 4.8). Technologies such as wavelength-switches coupled with ultra-long-haul optics can reduce the need for such transponders [79]; which can

† At the time of this writing, the 5400 system lists in its spec-sheet a power consumption of 2.7kW. However this is only if the entire 1.2Tbps system was dedicated to circuit-switching. Since we use both packet and circuit switching capabilities in equal measure, we revised the power consumption number to reflect the hybrid nature.

further help lower costs in all designs; but especially in IP-and-DCS design as we ask for very-long circuits in our full mesh topology (eg. LA to Boston, Seattle to Orlando).

OAM&P costs: Operations, Administration, Maintenance and Provisioning (OAM&P) is a broad term used to categorize the day-to-day functions involved in operating a network [103]. Through our talks with industry [76], we have allocated on average the number of man-hours per week dedicated to a type of equipment (Table 4.9). And we have assumed an hourly-rate of $25/hour for a network-operator [104].

Design	Router/Sw OAMP	WDM_OAMP	Transponder_OAMP	Total
IP over WDM	$1,497,600.00	$305,500.00	$4,546,750.00	$6,349,850.00
Optical Bypass	$1,372,800.00	$283,400.00	$4,312,750.00	$5,968,950.00
IP and DCS	$686,400.00	$239,200.00	$3,358,550.00	$4,284,150.00

Table 4.11: Annual Labor Costs for OAMP

Table 4.11 shows the annual-labor-costs for OAMP functions performed on the equipment in Table 4.8. But it is worth noting that the costs are more reflective of the work required for *maintenance* of the equipment, and not all OAMP functions (especially provisioning). As we mentioned at the start of this section, we cannot access proprietary labor-costs for all OAMP functions; especially the *hidden-costs* of manual coordination between IP and transport teams, where we believe our unified-control architecture provides the greatest benefits.

Rack Rentals: Rental space required for racks (in PoPs) is calculated on the basis of the size of the equipment (Table 4.9). As discussed earlier, the 1.28Tbps Backbone Router solution requires 2 racks, one each for the single-shelf-systems, and a third rack for the fabric-card-chassis. The Packet-Optical switch needs a single 19" or 23" rack. WDM equipment is much smaller and multiple such boxes can fit in rack. The difference between different rack sizes has been ignored. We were unable to obtain rack-rental rates for networking-equipment. So we emulate such rates with rental-rates for server-racks. We have assumed a monthly rental rate of $1500 [105]. Table 4.12 shows annual rental costs for the 3 designs.

Design	Router/Switch	WDM System	Transponder	Total
IP over WDM	$2,592,000.00	$705,000.00	$6,995,000.00	$10,292,000.00
Optical Bypass	$2,376,000.00	$654,000.00	$6,635,000.00	$9,665,000.00
IP and DCS	$792,000.00	$552,000.00	$5,167,000.00	$6,511,000.00

Table 4.12: Annual Rack-Rental Costs

Overall the limited-Opex analysis shows 37% cost savings in our IP-and-DCS design compared to the IP-over-WDM reference design for the original traffic matrix. When we scale the traffic to 10Tbps, the savings increase to nearly 45% (Fig. 4.29).

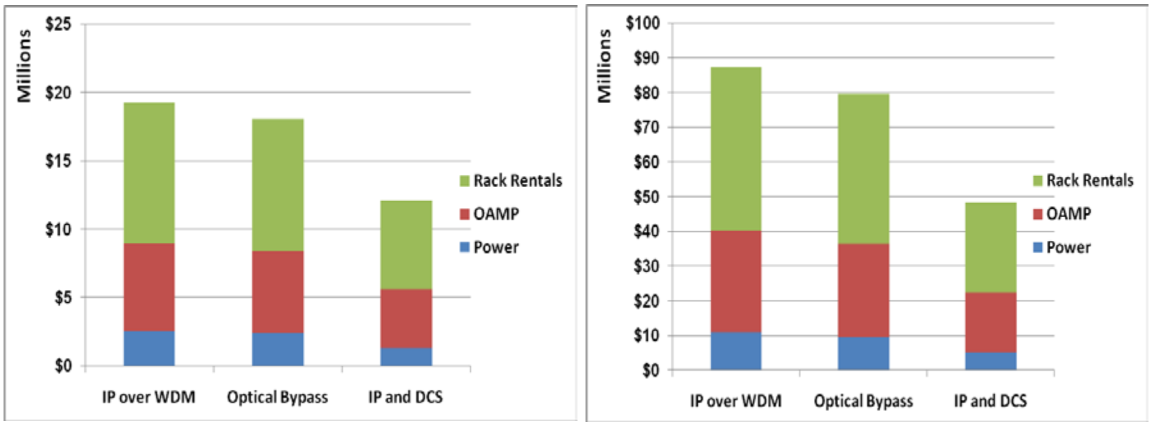


Figure 4.29: Opex Analysis for original (1X) & scaled (5X)Traffic Matrix

4.3 Summary

In this chapter, we explored the impact of our unified control architecture on IP core network design. Our objective was to determine ballpark numbers for the savings afforded by our architecture on the Total Cost of Operations (TCO – i.e. Capex and Opex) incurred by a carrier, for a large US continental IP network. We first considered industry standard practices for IP network design. These include IP-over-WDM and as a small-variation, IP-over-WDM with Optical Bypass. Then we proposed our converged packet-circuit network design and compared it to the other designs. While we model the transport network for each design case, we do so only in support of the IP network.

We defined a design methodology for IP-over-WDM networks, which includes dimensioning the IP network for demand, recovery, and traffic-uncertainties; and performed a Capex analysis for costs associated with all components of such design (ARs, BRs, WDM systems/transponders). As prior works of others have shown, the largest chunk of Capex comes from core-facing BR ports, due in large part to the high volume of transit traffic handled by BRs. Next we modeled the use of static-optical-bypass, as a solution for reducing transit-traffic in BRs. We showed that such methods reduce Capex by 10-15%, but ultimately run into diminishing returns as more bypass is introduced. The static nature of the bypass does not allow the solution to deal effectively with failures and varying traffic-matrices.

And so we proposed an IP core network that is as much dynamic-circuit-switched as it is packet-switched. The main characteristics of our IP-and-DCS network include: a) replacement of BRs in PoPs with Backbone Packet-Optical Switches (BPOS); b) the use of a full-mesh of variable-bandwidth circuits between core-PoPs; and c) the adoption of our unified control architecture for common control of packet and circuit switching.

We showed that such design choices help simplify the data plane; ease routing burden on the control-plane; and eliminate all transit-traffic in normal and recovery conditions; while maintaining the same level of over-provisioning in the network. We showed that we can save nearly 60% on Capex costs; such savings are insensitive to varying traffic-matrices; and scale better as the overall traffic grows to five times the original aggregate. Additionally, in a limited Opex analysis that considers power-consumption, rack-rental costs, and network-technician man-hours for maintenance, we showed nearly 40% in cost-savings. Additional Opex benefits come from the use of our unified control plane, which removes the need for manual-coordination between IP and transport teams for service provisioning.

In conclusion, we validated our claim that circuit-switching can make the Internet core more efficient if the two networks work together under common control, as the former helps the latter scale better to meet growing traffic demands.

Chapter 5

Introducing SDN Control in MPLS Networks

MPLS networks have evolved over the last 10-15 years and have become critically important for ISPs. MPLS is primarily used in two ways: to perform traffic engineering in IP networks, providing greater determinism and more efficient usage of network resources; and for enabling MPLS based L2 or L3 enterprise Virtual Private Network (VPN) services, which continues to be one of more profitable services offered by ISPs. MPLS is the preferred solution for such services, mainly because plain-vanilla IP networks are incapable of providing the same services; and older ways of providing these services using ATM or Frame-Relay networks are no longer used.

As carriers deploy MPLS they find that (a) even though the MPLS data plane was meant to be simple, vendors support MPLS as an additional *feature* on complex, energy hogging, expensive core routers; [†] and (b) the IP/MPLS control plane has become exceedingly complex with a wide variety of protocols tightly intertwined with the associated data-plane mechanisms.

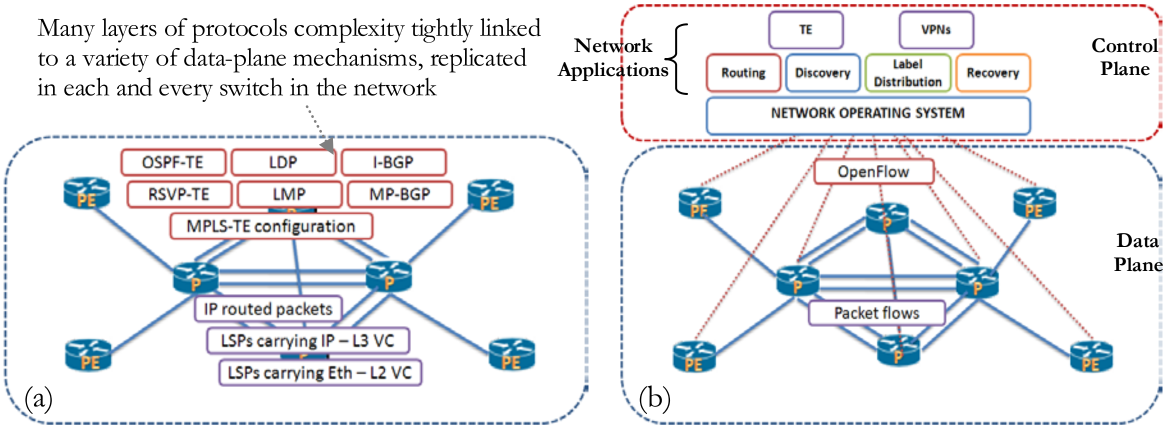
We make the observation that in any MPLS network, there are simple data-plane mechanisms of pushing-on, swapping, and popping-off MPLS labels in a label-switched-path. These mechanisms are controlled by a number of control-plane protocols that help

[†] eg. Cisco's CRS-1 and Juniper's T-640

provide the features and services (Fig, 5.1a). However, as we have shown in Chapter 3, any change to these services or the creation of a new service, in-most-cases, involves changes to these protocols or the creation of an entirely new protocol, which are lengthy, time-consuming processes. Yet, the data plane mechanisms remain the same simple push, swap, and pop operations.

And so in this work, we take a different approach to MPLS networks (Fig. 5.1b). We use the standard MPLS data-plane together with a simple and extensible control-plane based on OpenFlow and SDN. We find that the MPLS data-plane has similarities to the flow-abstraction. *But the MPLS control-plane does not make use of the map-abstraction*. Thus we retain the standard MPLS data-plane, and introduce the map-abstraction for the control plane.

There are significant advantages to doing so. The control-plane is greatly simplified and is de-coupled from a simple data-plane. We can still provide all the services that MPLS networks provide today; but more importantly, we can go beyond what MPLS provides today. We can globally optimize the services; make them more dynamic; or create new services by simply programming networking applications on top of the map abstraction. New capabilities are no longer tied to layers of protocols (which are eliminated). And the switch-API (OpenFlow) doesn't need to change either for all it gives is control over the simple push/swap/pop data-plane operations, which remain the same.



Many layers of protocols complexity tightly linked to a variety of data-plane mechanisms, replicated in each and every switch in the network

Fig. 5.1 (a) Today's IP/MPLS networks (b) MPLS with the Map-abstraction

In this chapter, we first discuss MPLS in relation to the flow and map abstractions. We show how flows in MPLS are close to the description of the flow-abstraction in the SDN context. And we also show how maps in MPLS networks are not quite the same as maps in SDN. But more importantly, we discuss how the map-abstraction is missing in MPLS. Accordingly all the benefits of the map-abstraction in terms of simplicity, extensibility and global-optimization are also missing in MPLS networks. We give examples of the benefits of introducing the map abstraction in MPLS networks.

To demonstrate that our approach can replicate services provided by MPLS today, we discuss a prototype implementation of MPLS based traffic engineering (MPLS-TE). There are two goals of this prototype: First, we show the simplicity of our approach compared to the existing MPLS control plane, by implementing nearly all the features that are part of MPLS-TE in just 4500 lines of code, compared with an estimated 100,000+ lines of code from a more traditional approach; and second, we show how the implementation of features can be made different from the traditional implementation in ways that either, greatly simplifies the feature, or provides a feature that MPLS-TE cannot provide.

Finally, we discuss how introducing the map-abstraction in MPLS networks fits well with our unified-control architecture for packet and circuit networks; a fact that makes our control architecture ideal for multi-layer networks.

5.1 MPLS Usage Models

There are two basic usage-models for MPLS, which correspond to the two services that MPLS provides – VPNs and TE. VPNs use what can best be described as a datagram model, similar to IP datagram forwarding. On the other hand, TE uses a flow-based model. Both models can and do exist simultaneously in today’s networks. In this section, we briefly discuss these models. Note that the terminology used for naming the models is not industry-standard. In fact the industry does not have terminology for the usage models. We introduce the terminology to best reflect the mechanisms involved.

5.1.1 MPLS Datagram Model

The datagram-like usage of MPLS is very similar to plain-vanilla IP routing. In this case:

- Label distribution happens in an *unsolicited* way [106], where every router sends out label-bindings for IGP-learned prefixes to each of its neighbors without being asked for it. For example, in Fig. 5.2, once the IGP (like OSPF or IS-IS) converges, every router learns of the router-addresses for all other routers in the IGP's domain. Each router creates label-bindings for these addresses, and then distributes these bindings to their neighbors (actually to their LDP neighbors). For example, in the figure, R6 receives label-bindings for R3's router-address from each of its neighbors R2, R4 and R5. Each binding implies that if R6 wants to reach R3 via a neighbor, it should use the associated binding advertized by the neighbor. R6 needs to use label 67 to go through R2; label 10024 through R4, and so on.

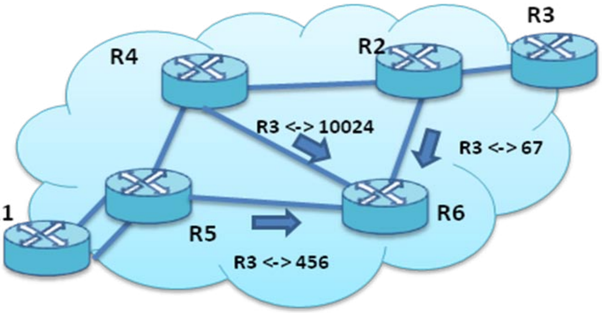


Fig. 5.2 Unsolicited Label Distribution

- The packets that match an FEC[†] at a router are then forwarded to the next-hop by inserting the label advertized by the next-hop. But the *selection of the next-hop* is still based on an SPF calculation from IP routing at each router. In other words paths for LSPs through the network are chosen by IP routing and thus are the same as IP shortest paths.

[†] FEC stands for Forwarding Equivalence Class – basically all packets that match an FEC (rule) are forwarded the same way (equivalently). The most common FEC is the IP-destination address.

And so, just like IP routing, each router makes an *independent decision* on what to do with incoming packets. In Fig. 5.3, R1 chooses to have two different flow definitions or FECs – one considers both IP src and dst addresses; the other defines the FEC to be all web traffic (TCP port 80) destined to an IP dst address range. Let's assume that both flows need to reach R3 to get forwarded to their ultimate destination. In this type of MPLS network, R1 *cannot define different paths (or LSPs)* for the two FECs from R1 to R3. The path that gets selected for the LSP is the IP-shortest path (say R1-R5-R4-R2-R3).

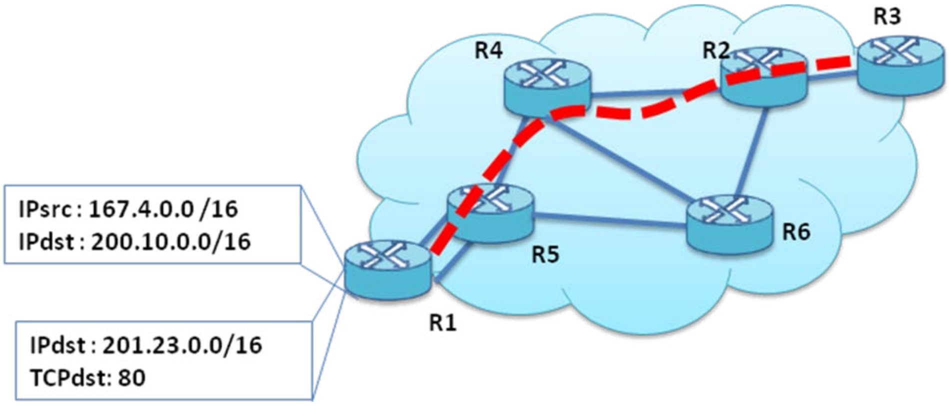


Fig. 5.3 MPLS Datagram Model

This is because the FEC definitions are not propagated to every router in the network. So when packets for the two different flows arrive at R5, the latter is unaware of how R1 defined the FECs. It simply considers *the only FEC that every router considers* for every packet – IP destination prefixes. R5 finds that both 200.10.0.0/16 and 201.23.0.0/ 16 need to go through R3; and that R4 is the next-hop (instead of R6) on the shortest IGP path to R3; finds the label-binding that R4 advertized for R3, swaps it on to the packet and sends it to R4. This procedure is repeated at R4 and R2. In other words it makes no sense to have different flow definitions in R1[†], because the flows differentiation is lost to the rest of the network, which treats them as the same. In other words, in this usage-model, an LSP is not actually *set-up*. The label-switched path develops 'on-its-own', and no router in the network has any control over the path the LSP takes through the network.

[†] Unless it is used for access-control into the network (accept, deny, rate-control). Again this is only at the edge of the network.

In essence, this usage model is simply IP-routing-with-labels, and is a holdover from the days when it was faster to do a lookup on a fixed 20-bit label than a variable length IP addresses. Today, it has found use in MPLS-VPNs, where the MPLS label-stack is used to deliver full mesh connectivity with address-space isolation. The outer label identifies the destination-router (PE) and the inner label identifies the VPN VRF instance. The path to the PE router is determined via IP routing, and forwarding is performed using the outer-label. We will discuss this MPLS use-case in the SDN context in Sec. 5.4.

5.1.2 MPLS Flow Model

In the flow-based usage-model, LSPs are actually set-up in the network by head-end routers (also called Label Edge Routers). Packets are classified into FECs. All packets in the same FEC are forwarded equivalently via an LSP whose path can be determined independent from regular IP routing (Fig. 5.4). Path calculation is done dynamically by the head-end router or offline by a management system. And it is typically signaled explicitly from the head-end to all other routers (LSRs) along the path using a signaling protocol like RSVP. Label distribution is therefore typically downstream-on-demand and ordered [106]. Importantly in this usage model LSRs do not make independent routing/ forwarding decisions on packets. Once an LSP is set-up and the LER has mapped an FEC to an LSP, packets are forwarded as part of *flows* (FEC+LSP).

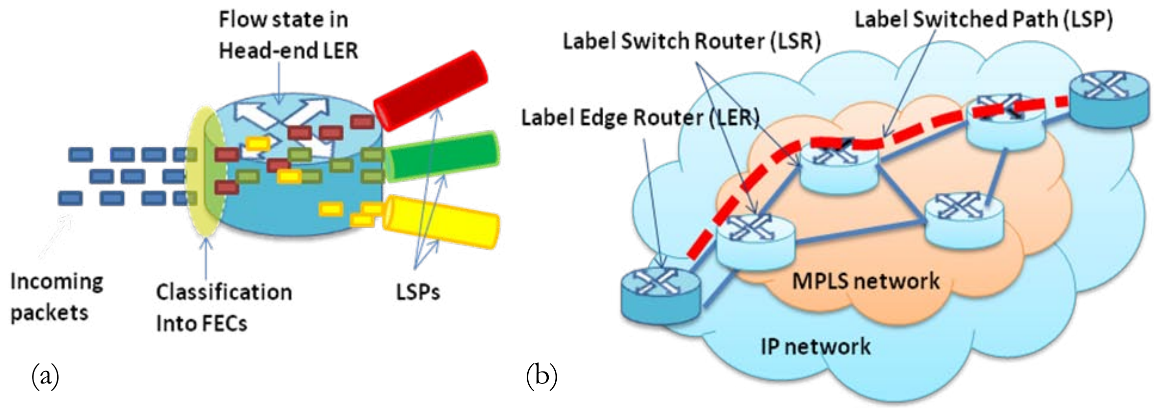


Fig. 5.4 MPLS Flow Model

The MPLS flow-model is used to provide MLPS based Traffic Engineering. Carriers widely use MPLS-TE today to achieve a) more deterministic behavior in IP networks; and b) greater efficiency in the utilization of network resources. In MPLS-TE the LSPs are more commonly known as TE-LSPs or tunnels. Traffic engineering is accomplished by routing the tunnels over under-utilized links in the network, and then routing IP traffic through those tunnels. MPLS-TE provides admission-control for TE-LSPs (tunnels) via bandwidth-reservation and constrained SPF routing. Additionally, there are a number of 'features' that MPLS-TE provides to ease the management, operation, and utility of tunnels. Such features include: Auto-Route, Auto-Bandwidth, tunnel-priorities, DS-TE, load-balancing, explicit-routes, re-optimization timers and so on. We will discuss this use-case in the SDN context in more detail in this chapter.

5.2 MPLS and Abstractions

The primary objective of this section is to show how MPLS compares to our control architecture. We show that while the MPLS definition of flows is not exactly the same as our definition, it is close enough to be applicable. But while MPLS retains and makes use of *a map*, it completely misses out on the map-*abstraction*. We discuss a few examples of what the map-abstraction brings to MPLS networks.

5.2.1 MPLS and the Flow-Abstraction

We mentioned before that MPLS has the concept of flows. In the following discussion we compare MPLS based-flows to the SDN based flow-abstraction from Sec 2.1.1. We conclude that MPLS based flows (FEC+LSP) are not as generic and flexible as the SDN flow-abstraction in terms of the 'match'-definitions and forwarding-actions. Neither do they support vendor-neutral switch-APIs.[†] But nonetheless, flows exist in MPLS based WANs.

[†] For example, the API into the FEC table is called Policy Based Routing (PBR) in Cisco routers, and Filter Based Forwarding in Juniper routers. Naturally the CLI commands are different.

- Logical Association:
  - SDN: A packet-flow is a logical association between packets that are part of the same communication and are given the same treatment in the network (within a switch and from switch-to-switch)
  - MPLS: MPLS has the concept of FECs (Forwarding Equivalence Classes) whereby packets are classified into FECs and forwarded equivalently in the switch and in the network via Label Switched Paths (LSPs). FECs therefore embody the logical association between packets in the same flow.
- Data Abstraction:
  - SDN: The data-abstraction is the representation of packet-switches as flow-tables, ports and queues. The flow is defined in tables which have the ability to identify the flow generically from multiple parts of the packet header in *each* switch.
  - MPLS: The data-abstraction differs from switch to switch in MPLS. In a Label Edge Router (LER) where the FEC is established, packets can be generically classified from multiple parts of the packet header. Such classification can be performed by techniques such as Policy Based Routing (PBR) working on a data-abstraction of a routing/forwarding table (RIB/FIB).[†] Once packets have been classified, a label is pushed on to the packet. Thereafter in all other switches along the flow's path, the Label Switch Routers (LSRs) have a data-abstraction of a label-table (LIB/LFIB) i.e. they only match on the MPLS label to identify the flow.

This distinction between the data-abstractions in LERs and LSRs can be useful; Using simpler exact-match forwarding on a label takes less space in forwarding tables than matching on packets generically. But it is also more restrictive – only MPLS labels can be matched with this data-abstraction. If however, a generic table abstraction is used in *every* switch along the flow's path, a) nothing prevents us from dividing things up like MPLS does by inserting an MPLS label; but b) if we wish we could do the same with other labels like VLAN tags; or c) in other cases, continue identifying

[†] Actually, PBR does not alter the routing tables. ASICs are implemented such that PBR policies supersede the forwarding decision in the routing table. Nevertheless, for the sake of comparison, we assume that the data-abstraction of the routing table in LERs is altered by an API such as PBR.

flows generically from multiple parts of the packet header, while *changing* the definition of the flow as we go along a path [54]. Thus our definition of the switch's data-abstraction is more flexible, which when coupled with the map-abstraction, makes networks more programmable.

- <u>Treatment within a switch</u>:
  - SDN: Packets that match a flow are treated the same way within a switch, where the flow-table applies the same set of actions on the packets.
  - MPLS: All packets that are part of the same FEC are certainly treated the same way within a switch. But again the set of actions that can be applied are more restrictive – pushing, swapping and popping off an MPLS label.
- <u>Treatment from switch to switch</u>:
  - SDN: Each switch through which the flow traverses does not make an independent isolated decision on the treatment to the same flow.
  - MPLS: This holds true when LSPs are established by head-end LERs.
- <u>Accounting & Resource Management</u>:
  - SDN: The flow-definition serves as a common-handle with which accounting and resource management can be performed at the flow-level in the network.
  - MPLS: This holds true for MPLS when using LSPs for traffic engineering.
- <u>Layer-independent Switch-API</u>:
  - SDN: The data-abstractions in all switches are manipulated by a layer-independent switch-API (like OpenFlow).
  - MPLS: Manipulation of the FEC in LERs may be exposed through the CLI by features like Policy Based Routing (PBR). Or it may be restrictive and hidden by features like MPLS AutoRoute [92]. In LSRs, the API is completely hidden and only operated on by label-distribution protocols like LDP and RSVP.

It is also worth pointing out that we compared the SDN flow-abstraction to the flow-based usage of MPLS (Sec. 5.1.2) where: multiple FEC definitions are possible; head-end routers make decisions on LSP paths; label-distribution is downstream on-demand and ordered; and resource management can be performed on the basis of LSPs [92]. MPLS based flows are more restrictive than the SDN flow-abstraction; but it is fair to say that flow-abstraction exists in some form in MPLS networks.

We *do not compare* the SDN flow-abstraction to the MPLS datagram-based usage-model (Sec. 5.1.1) for the following reasons: In the datagram-model, the only logical association (or FEC or flow-definition) that matters network-wide is the IP-destination address; forwarding decisions are still made independently router to router; and one cannot perform resource management at the flow level. However, even though the SDN flow-abstraction cannot (and should not) be compared to the MPLS datagram usage-model; the former can still provide the service the latter enables – VPNs. We discuss this point in more detail in Sec. 5.4.

5.2.2 MPLS and the Map-Abstraction

When comparing MPLS network control with our control-architecture, it is important to distinguish between the *map* and the *map-abstraction*. We discuss them individually.

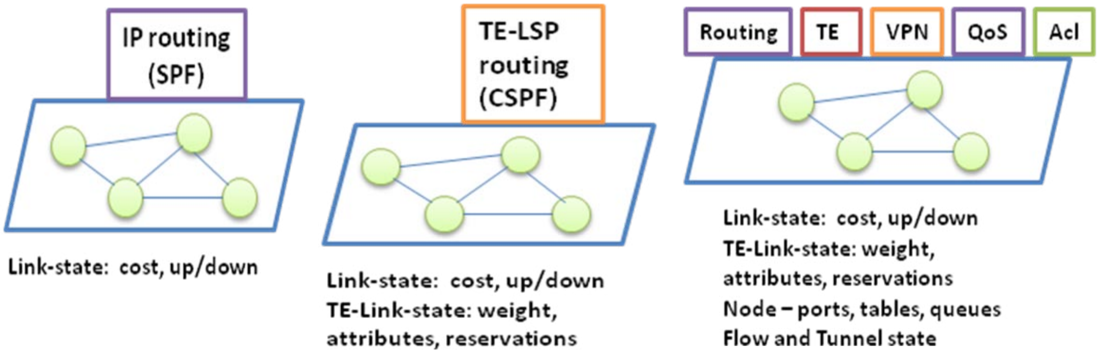

IP routing (SPF)
Link-state: cost, up/down
TE-LSP routing (CSPF)
Link-state: cost, up/down
TE-Link-state: weight, attributes, reservations
Routing
TE
VPN
QoS
Acl
Link-state: cost, up/down
TE-Link-state: weight, attributes, reservations
Node – ports, tables, queues
Flow and Tunnel state

Fig. 5.5 Network Maps in a) IP networks b) MPLS-TE networks c) SDN

Maps: A map by our definition is an annotated graph of the network topology. First let’s see how the maps themselves differ in different networks. In plain-vanilla IP networks, distributed routing protocols like OSPF and IS-IS distribute link state to each router in the network. This allows each router in the network to build a map (Fig. 5.5a) of their own that is consistent with maps built by other routers. The map includes topology information like nodes and links, as well as the 'cost' of each link for use in an SPF calculation. The only other link-'state' that is distributed by the routing protocol is whether the link is up-or-down. This is a very limited map, which allows the basic network-functions of SPF routing and re-routing around failures, and not much more.

In MPLS-TE networks, the same distributed routing protocols are used, but this time with TE extensions that enable it to carry more link-state. Such state includes maximum reservable bandwidth on a link, un-reserved bandwidth per priority level, link attributes and administrative weights [92]. And so, the map that gets constructed in each router (Fig. 5.5b) has node and link information as before, but now with additional link-state. This allows a router to perform more sophisticated path calculations that take into account bandwidth and other constraints on links when calculating TE-LSP paths.

An SDN map has all the information about the topology and link-state discussed so far. But in addition, it has a lot more information about the network-node internals like switch flow-tables, queues, ports and their state and statistics. It also has global information on not just link-state but also switch-state, and if needed TE-tunnel and packet-flow state (Fig. 5.5c, Fig. 1.10 & Sec. 2.2.1). The net result is that the map can be used to support not just traffic-engineering based features, but a host of other network-applications. Such applications may include access-control, mobility management, VPNs, bandwidth-on-demand, QoS, datacenter backup/migration, and more. In short, the map in SDNs, together with the flexible flow abstraction, makes networks more programmable.

Map-Abstraction: So far we have compared and discussed the differences in the maps themselves. Now let's consider the map *abstraction*. The map-abstraction helps centralize decision-making. It allows control-programs to be written in a centralized way with full, global visibility into the network, without worrying about how that global visibility (the map) is created, supported (perhaps with multiple physical controllers) and kept up to date. MPLS networks lack such a map-*abstraction*. We have discussed the benefits of the map-abstraction before (Sec. 1.4.2). Here we focus on the benefits in the context of MPLS networks.

Simplicity: The map abstraction helps simplify the implementation of control programs for the simple reason that they are no longer implemented as distributed applications. When control programs are centralized and decoupled from the process of state collection and dissemination, they do not have to contend with subtle interactions with the state distribution mechanisms; which can be complex and is often the source of many errors.

Take the simple case of a network-function like IP based traffic-engineering on top of plain-vanilla IP routing. Each router is a decision-maker and shortest-path decisions are made on the basis of costs assigned to links. In most cases routing-protocols assign link costs taking the bandwidth of a link into account. But network operators have the ability to configure and change the cost of any link, and often do so, to avail of some rudimentary IP traffic-engineering benefits (without invoking MPLS-TE and its complex control plane).

While changing link costs is simple, the effect it can have is far from trivial. Changing the cost of a link in one part of the network, potentially changes a lot of shortest-paths that can in-turn affect traffic in a completely different part of the network. In principle it can be disruptive to many (or all) traffic flows. Worse, while the routing-protocol converges, loops may get formed and packets may be lost. Thus changing link-costs is considered just as disruptive as link-failures. But when decision making is centralized and decoupled from the state-distribution mechanism, such a network-function can be a lot simpler. When link-costs change, the controller can simply be programmed to not re-route existing flows in the network, and use new shortest-paths only for new flows. Or it could selectively re-route existing flows. The choice lies with the application-programmer (network-operator).

As a more concrete example, in Sec. 5.3, we will show that centralization of control-functions results in simpler code. We have implemented nearly all the features of MPLS traffic engineering in fewer than 5000 lines of code [107], which is at least a couple of orders of magnitude smaller than distributed implementations of the same.

Reducing Protocol Load: The current IP/MPLS control plane uses a number of distributed-protocols that result in considerable computational load in router CPUs. Routers need to have all the intelligence necessary for aggregating and disseminating routing information, using distributed IP routing protocols like OSPF or IS-IS. In MPLS-TE networks, these routing protocols are extended to advertize extra information about links, and TE gives more reasons to generate control traffic as link-state changes more often[†]. Furthermore, another layer of complexity is added with the need for distributed signaling /label distribution mechanisms like RSVP-TE and LDP. And within a domain, an IP/MPLS network may additionally support a host of other protocols such as I-BGP, RIP, SNMP and MP-BGP, together with many more protocols for multicast and IPv6. All of these protocols contribute to control plane load, increased fragility and increased cost.

Some of these protocols are exceedingly complex due to multiple rounds of extensions over the years. RSVP is a good example. It was originally intended as an IntServ mechanism for hosts to request QoS from the network [112]. Intserv and RSVP were never used. But RSVP got extended to RSVP-TE to serve as a LSP signaling protocol. It is a poor choice for TE signaling; classic RSVP has many features (baggage) not intended for TE. For example, soft-state and frequent state-refresh messages were mechanisms for multicast-group support in classic-RSVP. But TE-LSPs don't involve hosts and are not one-to-many in practice. Further, RSVP runs directly over IP (instead of TCP); and so it needs its own reliable delivery mechanism. And it maintains multiple sessions per LSP. All of these 'features' result in lots of control plane messages and corresponding CPU load. Additionally, RSVP has been overburdened with many more features such as hierarchical LSP setup, point-to-multipoint LSP setup, LSP stitching, FRR setup, and GMPLS extensions, making an already complex protocol more bloated.

State-of-the-art routers come equipped with high-performance CPUs that can handle the computational load created by all these protocols. However carrier networks are not

[†] In our talks with Orange Telecom, we found that concern for "too many control messages" led them to turn off TE extensions on protocols like OSPF and IS-IS. They still use MPLS-TE but in a very static way.

all upgraded at the same time. Most often there is a mix of older and newer equipment. And from our talks with carriers[†], it is a real concern that the older ones cannot keep up. Consider this cycle –the carrier would have a need for new network functionality; they would approach router vendors to come-up with a solution; router-vendors would upgrade router-software (and sometimes change the protocols); upgraded software would in-turn increase CPU load; older routers/CPU cards would find it difficult to keep up; ultimately requiring an upgrade in router hardware.

SDN reduces switch CPU load by eliminating the need for many of these protocols. All current-distributed-protocol functionally can be replaced by a combination of network-applications, global-view and switch-API (OpenFlow). Furthermore, a Controller built on multiple-commodity servers can have much more CPU computational power that any one router. And in an SDN based network, when a carrier has a need for new network functionality, it need not result in upgrading switch-software or changing the switch-API. Our implementation of traffic-engineering functionality (in Sec. 5.3) is the perfect example of introducing new network functionality, simply by changing/ creating network-applications on top of the map-abstraction.

Extensibility: The map-abstraction not only makes writing control-applications simpler, it also makes it easier to introduce new functionality into the network. Such new functionality could mean new applications or changes to existing applications. Importantly, it does *not* involve changes to the state-distribution mechanisms. The latter is solved once, and abstracted away from the control programs.

This is obviously not the case in today's IP/MPLS networks, where network functions are implemented as distributed applications, and each-function ends up using its own state-distribution mechanism (Fig. 5.6). Consider the path for introducing new functionality in the network today: First the carrier would have a need; then it would have to ask vendors to implement a solution; vendors would take considerable time to come up with pre-standard solutions that don't interoperate; then they would debate in the standards bodies to come up with an interoperable solution; the vendors then may or may

[†] Google and Global-Crossing.

not implement the entire standard; then there are carrier lab trials, limited field trials and finally, if all goes well, deployment. This is a 5-10 year process. At any time during this innovation cycle, the original needs of the carrier may change, and thus extensions are born and outdated baggage is carried cycle after cycle.

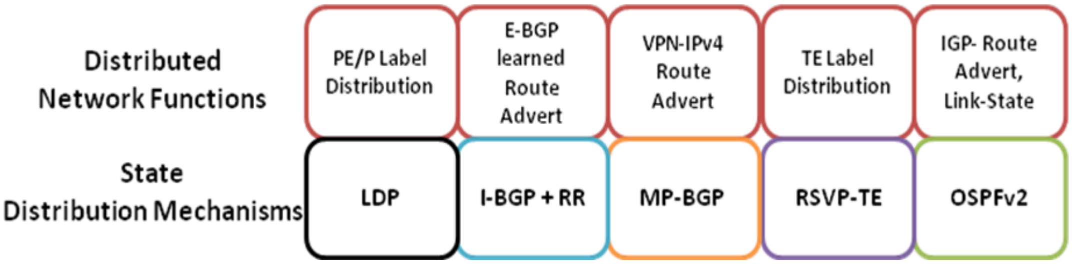

Distributed Network Functions
PE/P Label Distribution
E-BGP learned Route Advert
VPN-IPv4 Route Advert
TE Label Distribution
IGP- Route Advert, Link-State
State Distribution Mechanisms
LDP
I-BGP + RR
MP-BGP
RSVP-TE
OSPFv2

Fig. 5.6 Network Functions in Today's IP/MPLS Networks

SDN helps shorten the innovation cycle by making the network more extensible. It does so by un-chaining network functions/features/services from a) the creation of a new protocol; or b) the extension of an existing one; and c) requiring its implementation and deployment in each switch in the network

Global-View: One of the greatest benefits of the map-abstraction is the global-view it affords of all network-state. Such views allow the implementation of network-functions that globally-optimize network performance according to design or service needs. Without global-view, MPLS routers today can at best perform local-optimizations; which when coupled with the distribution-mechanisms, can lead to undesirable churn in the network. Consider an example from MPLS-TE.

Better utilization of network resources is one of the primary goals of traffic-engineering. MPLS performs traffic-engineering by forcing traffic through tunnels which are themselves routed over under-utilized parts of the network. The route that a tunnel takes depends on the bandwidth-reservation of the tunnel and the available un-reserved bandwidth on network links. As more tunnels get routed over a link (and therefore reserve bandwidth on it), the un-reserved bandwidth on the link diminishes, forcing newer tunnels to find routes over other links.

However the tunnel's reserved bandwidth is usually an *estimate* of the potential usage of the tunnel, made by the network-operator. The estimate is based on the source and destination routers and the average traffic between them. But traffic-matrices vary over time in un-predictable ways. And so a given tunnel's reservation could be very different from its actual usage at a given time, leading to an un-optimized network. Router vendors get around this problem by performing a local-optimization called Auto-Bandwidth.

With Auto-Bandwidth, a router at the head-end of a tunnel periodically checks the bandwidth usage of the tunnel, and alters that bandwidth-reservation to match the usage. But changes in reservation result in changes of link-state, as the links over which the tunnel is routed see a change in un-reserved bandwidth. Naturally such link-state has to be propagated to all routers in the network (depending on flooding-thresholds). But to further complicate matters, un-reserved bandwidth can be accounted for at *eight* priority levels; and tunnels can be assigned priorities whereby a higher-priority tunnel can preempt a lower-priority one by forcing the latter to re-route over a different path. This means that, as Auto-Bandwidth changes a tunnel's bandwidth reservation, the tunnel may force lower-priority tunnels to re-route, or the tunnel itself may get re-routed. In turn, re-routed tunnels further change link-state; causing greater flooding of link-state-update messages; and possibly even-more preemption of lower-priority tunnels. This cumulative effect is known as network-churn, as router CPUs have to deal with a lot of change in network state, especially in large carrier networks with tens of thousands of LSPs[†]. It is easy to see that churn is disruptive and undesirable from a network operator's view-point.

But churn is a direct result of local optimizations (like Auto-Bandwidth) performed by multiple decision makers (tunnel head-ends). Each router is only aware of the tunnels that it originates, and to some extent, the tunnels that pass-through it. For all other tunnels the router is only aware of the aggregate bandwidth reserved by these tunnels on links. In other words, even thought the router builds a map giving global TE-*link state*, it only has a local-view of tunnel-state (or TE-LSP state). In SDN, the map-abstraction gives global-view of network-state, including all tunnel-routes and their current-reservations and

[†] Talks with Tier-1 ISPs suggest the use of 50k to 100k LSPs .

usages. And so an application can perform global-optimization of the tunnels while minimizing (even eliminating) network-churn.

Thus we find that IP/MPLS networks can benefit from the introduction of the map-abstraction in its control-plane. Existing control-functions and services can be simpler to implement and globally-optimize, while new functionality can be added more easily to the network. In the next section we show how traffic-engineering functionality can be added to a network supporting the standard MPLS data-plane and an SDN based control plane, *without using any* of the existing IP/MPLS control plane protocols.

5.3 MPLS-TE Features & Prototype

In this section we show how we can reproduce services existing in IP/MPLS networks today, and improve upon them in meaningful ways. We use the example of MPLS Traffic Engineering (MPLS-TE) and the wide-variety of TE ‘features’ that come with it.

For each feature that we have implemented in our prototype, we describe the way it is implemented in today’s networks; how that differs from the way we have implemented it our SDN-based network; and finally how our approach enables some capabilities (within TE) that are not possible with the traditional approach. Importantly, our prototype serves as a validation of our architectural claims of simplicity and extensibility.

It also exemplifies the benefits of implementing networking-applications in a centralized way with a global-view, which comes from the introduction of the map-abstraction in MPLS networks.

5.3.1 MPLS-TE Features

We describe the following features: LSP Admission-Control; Auto-Route; Load-Balancing; DiffServ-Aware-TE; Auto-Bandwidth and LSP Priorities;

Admission-Control: The MPLS control-plane performs traffic-engineering primarily via an admission-control mechanism for LSPs. It is in principle similar to call-admission-control in telephony networks. The basic process is as follows:

- In the control plane, all IP links are assigned a maximum reservable bandwidth. TE-LSPs (tunnels) reserve part of the bandwidth on the links over which they are routed.
- The route (or path) that an LSP takes can be determined dynamically by the head-end router by running a Constrained-SPF algorithm. For example, given the source and destination routers for a tunnel (TE-LSP), together with its bandwidth-reservation (and priority, affinity etc.), CSPF can determine a path through the network that satisfies all the constraints – for example, a path over links that can meet the desired bandwidth reservation. If multiple such paths exist, the shortest one is selected.
- If a path that meets all the constraints cannot be found, the tunnel is rejected (admission-control). More importantly, tunnels tend to get routed (engineered) over links that are less used. If few pre-existing tunnels are routed over a link; the greater the chance that the link can satisfy bandwidth constraints for a new tunnel. And so network-resources are better utilized, and traffic in the tunnels may encounter less congested links (both TE goals).
- Once a path has been determined by a head-end router, it is signaled to all LSRs along the chosen route. Because there are multiple decision-makers (multiple head-ends) *contention* for resources may happen. For example, a head-end could setup a tunnel, but information on the resources used up by that tunnel may not propagate in-time to all other head-ends in the network (due to flooding-thresholds). In the mean time, another head-end tries to reserve a tunnel that contends for the same resources along links that overlap with the first tunnel. Because the path-decision for the second tunnel may have been made with stale information, every router along the way checks to see if the bandwidth-reservation is possible on the adjacent downstream link. If it is not, then the second LSP is rejected (again admission-control), and the (second) head-

end router is forced to re-route the tunnel, hopefully by this time with updated link-state (disseminated by the routing protocol). Ultimately it is a tradeoff – contention for resources can be reduced, if TE link information is always rapidly disseminated and head-ends calculate paths with up-to-date state; but it also ends up causing a lot more control-plane messages and CPU load.

- It is important to note that bandwidth reservations and LSP admission-control are purely control plane concepts. MPLS-TE *does not require* enforcement of tunnel bandwidths in the data plane [92], because the primary objective is to steer traffic, not to police or shape it in the data-plane. The latter techniques can be used in addition to MPLS-TE if desired.

SDN based Admission Control: Our SDN based implementation takes advantage of the fact that admission-control of LSPs are handled purely in the control plane. The TE application written above the map-abstraction: allocates reservable-bandwidth to each IP link; performs CSPF routing of tunnels; sets-up the tunnels in the data-plane by using OpenFlow to distribute labels to switches along the LSP path; and updates the map with LSP-path information and un-reserved bandwidth information for each link along the path. Importantly, since the Controller (more correctly the TE application) is the *only decision-maker*, contention for resources does not happen in this scenario. The Controller maintains the network-map and keeps it up-to-date with all tunnels and link reservations. So when a new tunnel is requested by the network-operator or by another application, the Controller is completely aware of all network-state, and can perform admission-control and LSP-setup without any contention. The data-plane switches are blissfully unaware of any bandwidth reservations for the LSPs. As a result, the data-plane switches themselves do not perform any admission-checks.

Auto-Route: Once tunnels are up in the data-plane, traffic needs to be redirected from IP links into the tunnels by LERs. Auto-Route is an effective way to do this without incurring the scaling problems of exposing tunnels to distributed-routing-protocols.

This latter point can best be explained with Fig. 5.7. Assume packet-flows from R1 to R3 are routed via R5→R6→R2. When the TE tunnel is created between R5 and R2, the flows need to be re-directed through the tunnel, as the tunnel is now a single-hop path between R5 and R2 (as opposed to a minimum of 2 hops for every other path[†]). Note that Fig. 5.7a shows the tunnel 'conceptually' as a single-hop between R5 and R2. But physically the tunnel is routed over the IP links in the network. For example, Fig. 5.9b shows one possible path the LSP takes in the IP network. Despite the number of hops in the physical tunnel path, from the perspective of routing packets in the IP network, the tunnel is still a single hop.

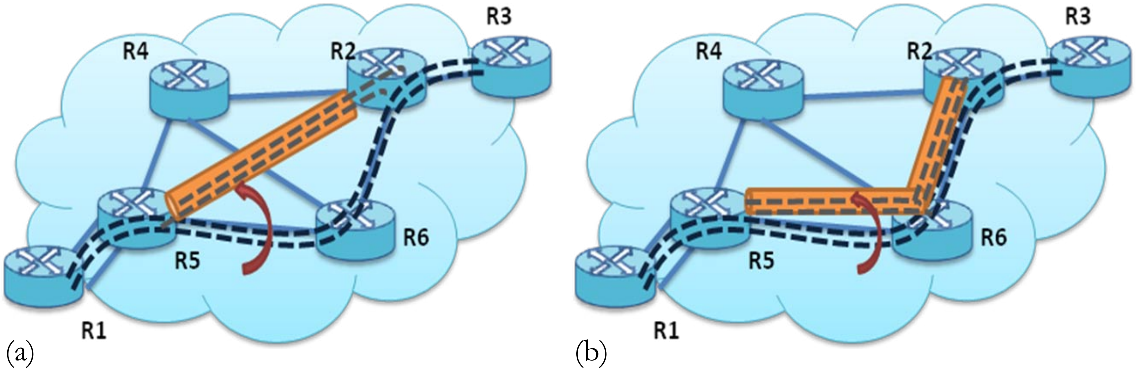


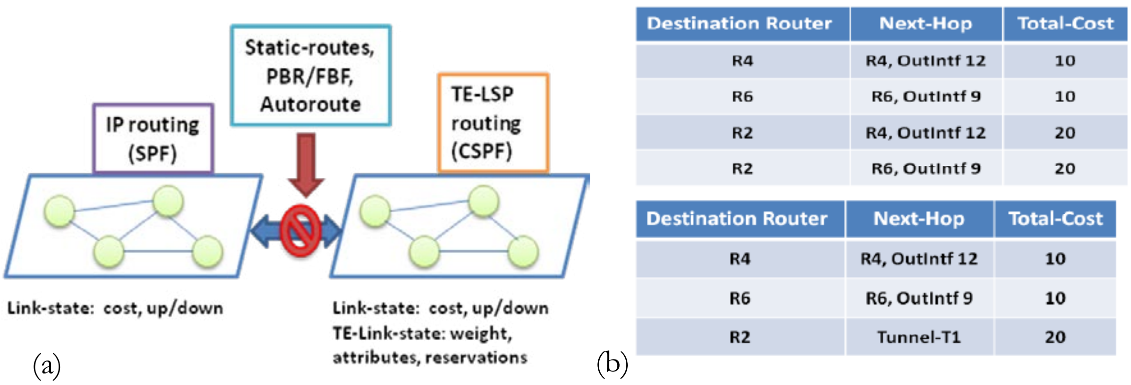
Fig. 5.7 Auto-Route: re-routing packets into a TE-LSP (tunnel). Tunnel is shown: (a) conceptually (b) physically routed over the IP links.

However this perspective does not materialize directly in the map over which packets are routed. The tunnels are *not* exposed to the distributed routing protocol i.e. routing adjacencies are not created across the tunnels. And so the map that the routing protocol creates in each router does *not* include the tunnels. The primary reason for not exposing tunnels to OSPF or IS-IS is the same $O(N^2)$ scaling issue discussed in the IP-over-ATM context in the previous chapter[*]. If the routing-protocol were exposed to the tunnels, then they would have to treat them as links and carry state for them, which is a scaling problem – as the number of links increases, more state needs to be disseminated; and when failures happen lots of control messages are generated, which in-turn increases router-CPU processing loads and could cause routing protocol instability.

[†] Assuming all links have the same IGP cost and the tunnel cost is that of a single-hop.

[*] See Sec. 4.2.1 under SDN based Unified Control Plane

But if tunnels are not allowed to show up in the IP-routing map, then how can packets be routed into them? This is where mechanisms like static-routes, PBR/FBF and Auto-Route play a role. Consider Fig. 5.8(a). MPLS-TE networks maintain *two* maps; one is used for regular SPF routing of IP packets; the other for CSPF routing of TE tunnels. All the mechanisms mentioned above can *change* the routing-decision made by SPF.



Destination Router	Next-Hop	Total-Cost
R4	R4, OutIntf 12	10
R6	R6, OutIntf 9	10
R2	R4, OutIntf 12	20
R2	R6, OutIntf 9	20

Destination Router	Next-Hop	Total-Cost
R4	R4, OutIntf 12	10
R6	R6, OutIntf 9	10
R2	Tunnel-T1	20

Fig. 5.8 (a) Need for Auto-Route (b) Effect of Auto-Route on Routing tables

Auto-Route, as the name suggests, re-routes packets into tunnels automatically [92]. It works by replacing routing decisions during SPF calculations. Consider the *partial* routing table shown in Fig. 5.10b calculated by R5 for the network shown in the upper part of Fig. 5.8b. It shows two equal-cost routes to R2 that have different next hops, R4 and R6; which R5 reaches via out-going interfaces 12 and 9 respectively. But invisible to SPF, R5 also has a tunnel to R2. Here is where Auto-Route kicks in (shown in the lower table in Fig. 5.8b) to reflect the effect of the tunnel. Both entries in the upper table are replaced with an entry where the next hop to R2 is the *tunnel* T1. The tunnel-interface is a logical (virtual) interface which is manifested behind-the-scenes by the physical interface (and label) on which the tunnel starts.

While Auto-Route is automated, it is also inflexible. It works very well for SPF routing based just on the destination-IP prefix. But it makes it hard to take a different (automated) approach to routing. For example, routing different types of traffic or different customer’s traffic differently is not possible with Auto-Route. If a different approach is required, then the network-operator has to fall back on static-routes and PBR, to supersede the decisions taken by SPF and Auto-Route. But while such mechanisms are flexible, they are not automated. They require human-intervention typically through the CLI in multiple routers along a path for each packet-flow that the operator may wish to route differently.

SDN approach to Auto-Route: SDN allows *both* flexibility and automation in its approach to routing. Together these attributes present a programmatic solution.

In contrast to Fig. 5.8a, with SDN we take the approach shown in Fig. 5.9. The network-map allows full visibility to node and link state. To this state (which comes from the switches), we add the link-state attributes we need for traffic engineering such as max-reservable-bandwidth, unreserved-bandwidth, priorities and other operator-defined attributes. TE routing of LSPs (tunnels) can be performed by an application on this map, and the resulting tunnels *are introduced as unidirectional links* in another map.

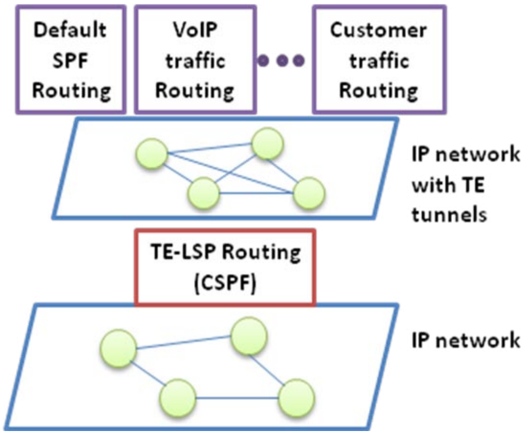


Fig. 5.9 Auto-Route in SDN

This latter map is used for routing packet-flows. We can simultaneously route different kinds of packet-flows differently – for example, traffic can be routed on the basis of the type of end-user application it supports (voip, web, video etc.); or it can be routed differently for different customers or services; all other traffic can use default IP

SPF routing. And the usage for each routing mechanism is programmatic, automated and dynamic.

We can take this approach because SDN *does not use* distributed link-state routing protocols to disseminate state; and so it is not limited by scalability issues of the same. In our implementation of Auto-Route (presented in the next section) the maps are updated by the Network-OS and switch-API (OpenFlow) which replace all the functionality of distributed routing protocols.

Load-Sharing: One of the direct consequences of Auto-Route is that it becomes harder to perform load-balancing in IP/MPLS networks. Especially for flows destined to the tail-end of a tunnel. Consider Fig. 5.7a. There are essentially three equal-cost routes from R5 to R2 - two that go via IP links and transit through R4 and R6, and the third is the tunnel from R5 to R2. Even assuming that the tunnel is routed on one of those paths (say R5→R6→R2), it still leaves us with two paths for load-balancing – the tunnel and the other path (R5→R4→R2). But due to Auto-Route (see lower table in 5.8b), only one of the paths – the tunnel – is used. In fact, due to the nature of Auto-Route, it is simply not possible to load-share traffic to a TE tunnel-tail between an IGP path (from SPF) and a TE tunnel-path [92]. The general solution to this problem is to create *multiple* tunnels to the same tunnel-tail and then load-share traffic between them. But the obvious drawback is that there are more TE-tunnels to create and manage.

SDN based Load-Sharing: Since we have more control over routing, and we represent tunnels as unidirectional links in the topology map, our approach to load-balancing is simple. In the above example, we simply route (and therefore load-share) traffic-flows over two paths – one that goes over the virtual-link that represents the tunnel, and the other over the IP links R5→ R4→R2. In our implementation we perform load-balancing on a flow-level (for example by matching on the TCP/IP 5-tuple) in the control plane. But this can easily be performed in the data-plane with the ‘select’ group-construct, which is part of version 1.1 of the OpenFlow specification. Here the controller can add the physical port on R5 that leads to R4, as well as the physical port and label that represents the start of the tunnel, to the same select-group to ensure load-sharing. Importantly we do not need two tunnels for load-sharing, and it does not matter if the two paths have the same cost or not.

DiffServ-Aware Traffic Engineering (DS-TE): DS-TE is a complicated marriage of two mechanisms: a) QoS based on the Differentiated Services architecture [115] that controls the per-hop behavior (PHB) of various traffic classes defined by the DiffServ Code Points (DSCP) bits; and b) MPSL-TE that controls the hop-to-hop (or path-level) behavior for IP traffic.

It is worth noting that on-their-own neither can *guarantee* QoS [92] on an end-to-end basis (within a network domain). For example, while DiffServ mechanisms can prioritize, shape and police traffic in class-based queues *within a switch*; it has no control over the path the traffic takes in the network domain. And so, congestion in a downstream node can cause traffic in some classes to suffer even with PHB control, as the congestion information is localized and not propagated back to upstream nodes. In this case, TE can help by trying to avoid congestion a-priori by steering traffic away via tunnels that reserve link-bandwidth and are admission-controlled.

However tunnel admission control is agnostic to the type (or class) of traffic that is being carried in the tunnel. And so traffic for a certain *class* of traffic cannot be steered away with regular TE. Therefore the purpose of DS-TE is to *enable tunnel admission-control on a per-QoS class basis*.

Consider Fig. 5.10a. It shows the queues that feed an outgoing port on which two (regular) MPLS-TE tunnels are routed. The TE tunnels reserve bandwidth on the link without taking into consideration the data-rates supported by the queues. Since the tunnels are unaware of the queues, they carry traffic for all class-types. Note that the highest-priority traffic in both tunnels comes out of the same queue. During times of congestion, it is possible that the min-rate guaranteed by the high-priority queue may not by enough to sustain all the high-priority traffic arriving at the queue.

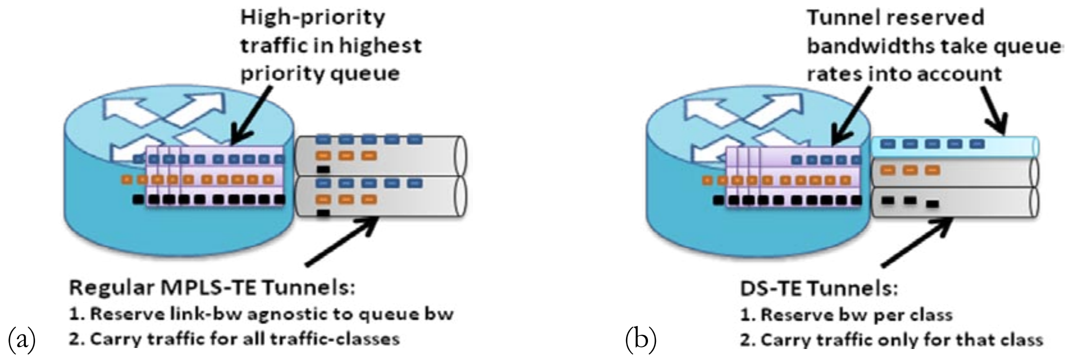


Fig. 5.10 (a) DiffServ with regular MPLS-TE (b) DS-TE

DS-TE tries to solve this problem by trying to steer away traffic for such a traffic-class (Fig. 5.10b). It does so by allowing tunnels to reserve bandwidth only for a certain class-of-traffic; out of a sub-pool of link-bandwidth that matches the queue-bandwidth for that class. For example, if the queue for the highest-priority traffic supports 100 Mbps, then tunnels for that class can only reserve link-bandwidth from the sub-pool of 100 Mbps, even if the max-reservable bandwidth may be 1 Gbps. This way if one tunnel for this class reserves 100 Mbps from the sub-pool, then other tunnels are not admitted on this link and thus forced to steer away from it.

SDN-based DS-TE: Diffserv-aware TE uses the same mechanisms as regular MPLS-TE; the subset of link-bandwidth advertized per-class, as well as per-class tunnel reservation and admission-control are all still purely control plane concepts. And so with SDN we can perform all of these tasks in the Controller. But the SDN based implementation has two distinct advantages:

- The process of forwarding class-specific traffic down a DS-TE tunnel is not trivial and hard to automate in networks that use DS-TE today. Auto-Route cannot help as the mechanism does not distinguish between traffic-classes. The only choice is the use of Policy Based Routing (PBR) [92], which requires configuration via manual-intervention and is not very programmatic. This is essentially the same issue as pointed out in the section on Auto-Route. With SDN and the use of OpenFlow, forwarding traffic-type (or class) specific packet-flows down a tunnel is flexible, automated and therefore programmatic.

- DS-TE currently allows the advertisement of only one subset (called sub-pool) of link-bandwidth, and so can be used for only one traffic-class [116]. Herein lies the power of SDN. If operators tomorrow wanted the ability to create traffic-type specific tunnels for not one but *two or more* classes of traffic, it would *require a change to the distributed routing protocol* to carry the advertisement of more bandwidth sub-pools (plus the associated upgrades to router-software). In SDN, no protocols would change. Only the CSPF and related applications in the controller need to be changed.

Auto-Bandwidth: In the previous section we discussed the Auto-Bandwidth feature and associated problems with local-optimizations and network-churn. Here we would like to focus on its implementation. Auto-Bandwidth is implemented in the routers at the head-end of TE tunnels on a per-tunnel basis. Periodically the router collects information on the actual bandwidth-usage of the tunnel (eg. every 5 mins). At a different periodicity, it alters the bandwidth-reservation for the tunnel based on the observed usage. The change in the reservation for the tunnel is signaled to all other routers in the LSP; it also changes link-state which is then disseminated to all other routers in the network by the routing-protocol.

SDN-based Auto-Bandwidth: In our implementation, since all tunnel reservations, link-state and admission-control is maintained and performed in the Controller; Auto-Bandwidth is performed as an application in the Controller as well. The only thing we need from the switches is information on the usage of the tunnel. OpenFlow defines an LSP simply as another flow (instead of a virtual-port). Thus we can obtain flow-statistics (like transmitted-bytes) from the switch via OpenFlow, to guage LSP usage. The application-programmer can set intervals for polling switches and adjusting the bandwidth-reservation. Changes in bandwidth-reservation are not signaled to the data-plane as the latter is unaware of any bandwidth reservations for the LSPs. Furthermore, changes to link-state that come about from the change in the tunnel reservation, requires only updating the annotated topology-map and not the switches. The only time that Auto-Bandwidth induced change needs to be propagated down to the data plane is when it causes one or more tunnels to re-route. In such cases, the Controller processes the change by removing flow-state for the old tunnel route and replacing it with the new tunnel-route in the affected LSRs. As we have mentioned before, this can cause significant churn in a large network with many tens-of-thousands of LSPs. But with SDN, the effect-of-churn is minimized as routers no longer have to deal with RSVP or OSPF state-update messages. They only need to process/generate OpenFlow messages that change flow (LSP) routes. In Sec. 5.4, we'll discuss a way in which Auto-Bandwidth can be eliminated altogether with SDN.

LSP-Priorities: One final MPLS-TE feature we cover is LSP priorities. TE-LSPs can reserve bandwidth at eight different priority levels (with 0 being the highest priority and 7 the lowest). Accordingly links advertize un-reserved bandwidth at each of the priority levels, for both the global pool and the solitary DS-TE sub-pool. LSP priorities are used for preemption – a higher priority LSP can force a lower priority LSP to re-route if there isn't enough un-reserved bandwidth available for both of them on a link they both share. For example, if a link has maximum reservable bandwidth of 900 Mbps; of which 500 Mbps has been reserved by a Priority-0 tunnel; and 300 Mbps by a Priority-1 tunnel; then 400Mbps of unreserved bandwidth is available at Priority-0, and only 100Mbps is available at Priorities 1-7. But if a new Priority-0 tunnel comes along reserving 200Mbps; then all three tunnels cannot co-exist on the link as their cumulative reservations exceed the maximum reservable bandwidth on the link. The lower priority-1 tunnel is forced to re-route – the head-end for this tunnel periodically tries to find a new path which satisfies all its constraints.

<u>SDN based LSP-Priorities:</u> As with all other LSP attributes, the LSP-priority is just a number maintained in the annotated topology map within the Controller. The data-plane switches have no information on LSP priorities. We maintain tunnel priorities in the

controller, as well as link-state for reservable bandwidth at eight priority levels. CSPF is performed in the Controller; tunnel-priorities are taken into consideration; and any resultant pre-emption is achieved using OpenFlow messages to add, delete or modify flow-entries in LSRs corresponding to TE tunnels.

5.3.2 MPLS-TE Prototype

In this section we discuss a prototype implementation of MPLS Traffic-Engineering in an SDN based network (MPLS-TE). There are two goals of this implementation: First, we demonstrate that our approach is capable of reproducing existing services offered by MPLS network today, while validating the simplicity of our approach compared to the existing IP/MPLS control plane; and second, as described in the previous section, we show how the implementation of the features can be made different from the traditional implementation in ways that either, greatly simplifies the feature or provides a feature that MPLS-TE cannot provide.

Data-Plane: At the time this work was done (late 2010), handling of MPLS labels was just being introduced into the OpenFlow protocol (version 1.1[117]). As a result, there weren't any OpenFlow enabled switches (or controllers) that supported MPLS data-plane capabilities via OpenFlow.

We added MPLS data-plane capabilities to a software switch called Open vSwitch (OVS[118]) that fully supported version 1.0 of the OpenFlow protocol. We also had to change the OpenFlow protocol (v1.0) to add the capability of controlling an MPLS data-plane. To accomplish the latter, we borrowed the MPLS related features from v1.1 of the protocol and added it to v1.0[†].

[†] We chose not to implement the entire v1.1 spec as it includes a number of other features like groups and multiple-tables, which were not essential to this experiment.

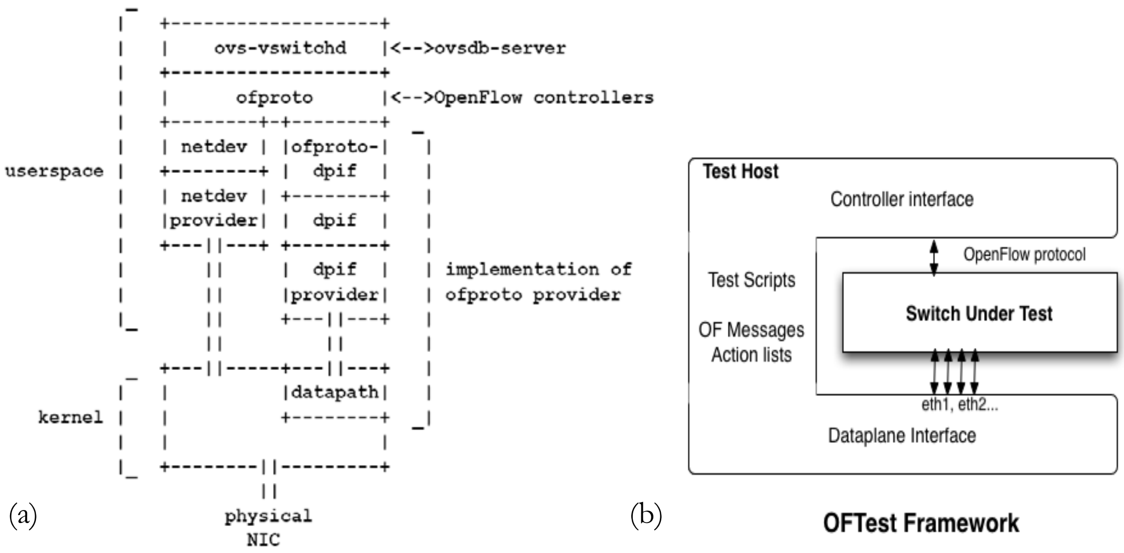


Fig. 5.11 (a) Open vSwitch Software Architecture [119] (b) OFTest Framework [120]

Fig. 5.11a shows the OVS software architecture. Details can be found in [119]. Here we focus on the parts we changed to add MPLS data-plane capabilities. OVS can be used purely as an OpenFlow switch, or it can be used with added functionality that is part of OVS. We chose to use it as an OpenFlow switch using the ovs-openflowd program instead of the ovs-vswitchd program. The former does not require communication with an ovsdb-server. Next in the stack shown in Fig. 5.11a, the ofproto module communicates with an external Controller. We changed this module to work with our additions to version 1.0 of the OpenFlow protocol borrowed from version 1.1 (for MPLS support). The ofproto module communicates with the switch using the ofproto-dpif interface.

The OVS switch datapath can be in kernel space (as shown in the figure) or in userspace. We chose the userspace datapath implemented in the dpif-netdev module, which also instantiates the datapath-interface (dpif) provider class. We added support for the MPLS data plane in the dpif-netdev module [120]. This included the ability to match on the outermost MPLS label (id and traffic-class bits) and perform the actions of pushing, changing and popping off multiple labels from packets together with TTL manipulations. Since our software switch was hosted in Linux, we used the built-in netdev-linux module that abstracts interaction between ofproto and Linux network-devices (like Ethernet ports).

We tested our implementation of the MPLS data-plane using the OFTest framework [121,122] shown in Fig.5.11b. OFTest includes a data-plane interface that allows sending and receiving packets to a switch-under-test (OVS in this case). It also includes a control-plane interface that implements a light-weight OpenFlow controller that can be used to interact with the switch-under-test using the OpenFlow protocol. In our case this controller communicates with the ofproto module in OVS. We used OFTest to write test-scripts that verified the correct functionality of the MPLS data plane.

Control-Plane: The control-plane was hosted in a server running NOX [15]. NOX was modified to include our changes to version 1.0 of the OpenFlow protocol, thereby enabling control of an MPLS data-plane [123].

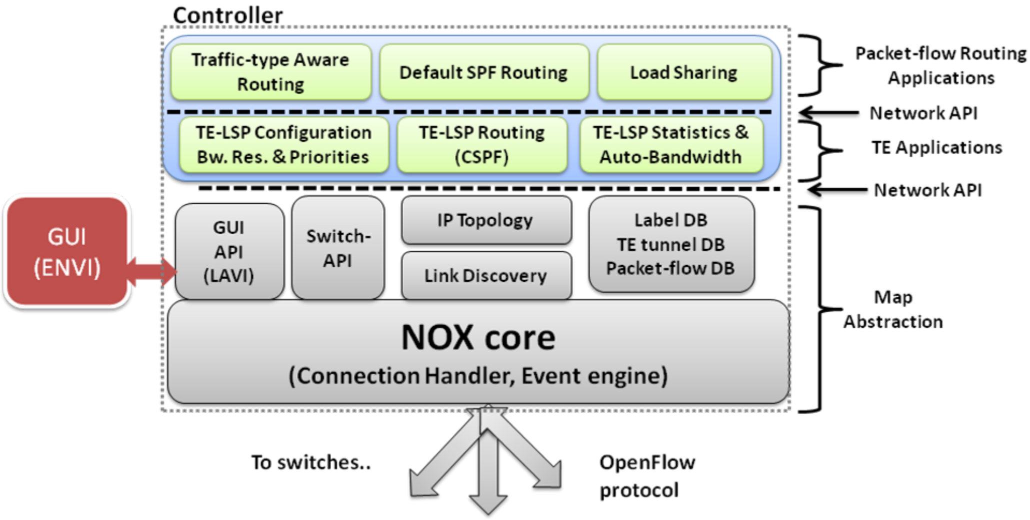


Fig. 5.12 MPLS-TE NOX Software Architecture

The software architecture is shown in Fig. 5.12. We use NOX's basic event engine and connection-handlers to the switches that are collectively referred to as nox-core. We also use NOX's link discovery module to discover IP links and construct the annotated topology-map. And we include several data-bases for packet-flows, tunnels and MPLS-label information. Together with the switch and network-APIs, these modules present applications with a map-abstraction.

The first set of network-applications involves TE functionality. The TE-LSP configuration module is responsible for reading tunnel configuration files or alternately responding to config-messages from a GUI. Tunnels are configured by specifying the head-end and tail-end routers; the desired bandwidth and tunnel priority; the type of traffic the tunnel is designated for; and whether Auto-Bandwidth is enabled on the tunnel or not. With this information the config module uses the TE-LSP routing module to find a path between head-end and tail-end routers that satisfy all the constraints (bandwidth, priority, traffic-class). Once a path is found, the config module uses the network-API to establish the tunnel in the data-plane by inserting flow-entries for the LSP which includes label information. If Auto-Bandwidth has been configured on the tunnel, it also activates the polling process for LSP usage information in the TE-LSP statistics module. This information is used by the Auto-Bandwidth module to periodically change the tunnel reservation and update the TE-LSP database. But before such an update happens, the Auto-Bandwidth module checks the CSPF module to make sure that the new-bandwidth reservation is possible along the existing tunnel path. If it is possible but requires re-routing lower priority tunnels or re-routing the tunnel itself, it processes such change by invoking methods in the network-API.

Together the TE applications create, manage, re-route, and tear-down TE-LSPs (tunnels) in the data-plane. As explained in the previous section, our map-abstraction allows us to create another map above the TE applications (see Fig. 5.9). This latter map represents the TE tunnels as unidirectional links, on which packet-flow routing can be performed in an automated and programmatic way. We create three routing-modules for packet-flow routing. The default SPF routing-module uses NOX's built-in all pairs shortest-path routing module. The only difference is that instead of using this module directly on the IP topology, we use it on the topology that includes the IP links as well as the tunnels represented as unidirectional links. The default routing module is used for all packet-flow routing. The other two modules are plugins into the default routing modules. The traffic-type aware routing module ensures that only those flows of a certain traffic-type (voice, video, web) are allowed into tunnels meant for the traffic-type (DS-TE). Essentially we use traffic-type to emulate traffic-classes (without actually setting or using the DSCP bits). And the load-balancing module ensures that traffic-flows can be load-shared between regular IP-links and links representing tunnels, to the tunnel-tail.

Experimental Setup: Our network of software switches is created in the Mininet environment [124]. Mininet is an open-source tool that creates a network of interconnected software switches on a single PC for emulating networks. It uses Linux process–based virtualization to run many hosts and switches on a single Linux kernel [125]. We use Mininet to emulate a wide-area IP/MPLS network. Our prototype system runs 15 instances of OVS (modified) to form the IP/MPLS network (Fig. 5.13).

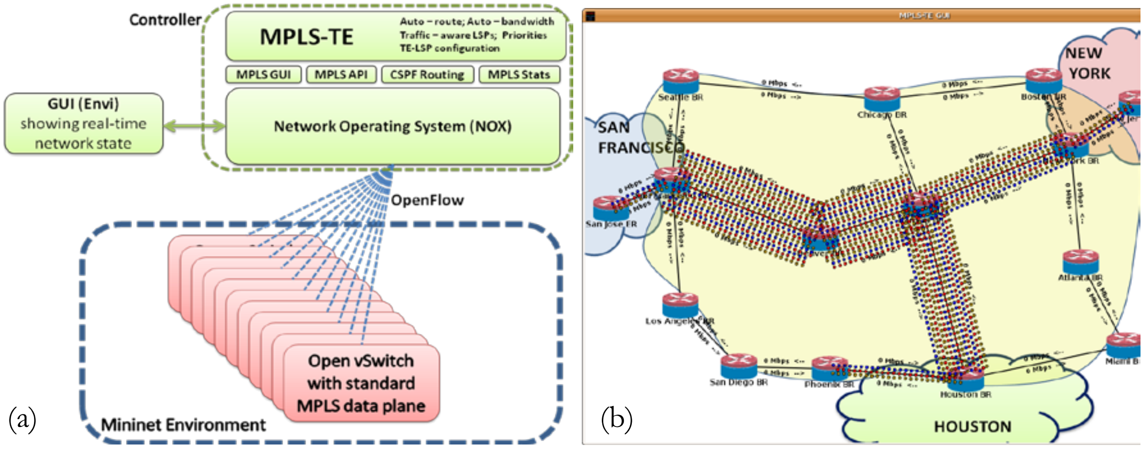


Fig. 5.13 MPLS-TE (a) Prototype (b) GUI view showing Emulated IP/MPLS WAN

Fig. 5.13b shows the GUI view for our emulated-WAN. The GUI shows network-state re-created in real-time via communications with the Controller. The emulated-network has nodes in 15 cities spanning the continental US. In the San Francisco (SFO) and New York (NYC) clouds, San Jose and New Jersey connect to the wide-area via the routers in SFO and NYC respectively. In Fig. 5.13b the GUI shows wide-area IP links between the cities. Unidirectional TE-tunnels are routed on these bi-directional IP links by reserving bandwidth on the links in the direction of the tunnel (from head-end to tail-end). The GUI displays reserved-bandwidth information for each link in both directions. Each link is assumed to be a GE link, with a maximum reservable bandwidth of 90% of the link-bandwidth (i.e 900 Mbps). Finally, Fig. 5.13 also shows traffic-flows for three different traffic types (voice, video and web traffic) originating from traffic-generators (not shown) in San Jose and New Jersey.

Experiments: In the previous section we discussed several features related to MPLS-TE. The primary goal of this experiment is to achieve the basic functionality of traffic-engineering itself. Secondary goals include the ability to show TE related features such as Auto-Bandwidth and DS-TE.

Tunnel-Routing and Packet-Flow Routing: Traffic Engineering in MPLS is a two-step process: a) admission-control of bandwidth-reserved TE-LSPs (or tunnels) performed during a CSPF routing process for the TE-LSP; and b) (Re-) Routing packet-flows through a newly created TE-LSP. In Fig. 5.13b, we see the operation of a network without traffic-engineering. Traffic flows from the SFO and NY areas, destined to Kansas, Phoenix, Houston and NY, all follow the shortest path in the network; potentially congesting the SFO ←→ DEN ←→ KAN←→NY links.

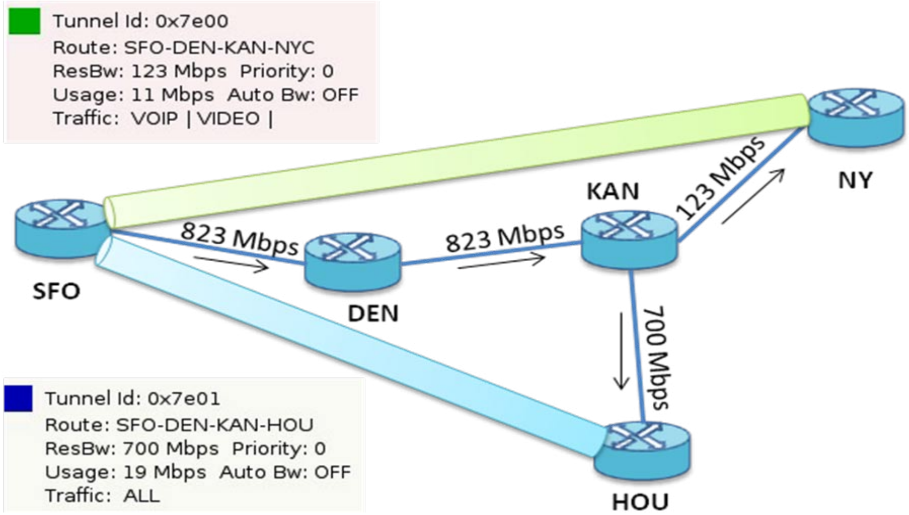


Fig. 5.14 Tunnel Routes, Bandwidth-Reservations & Auto-Route

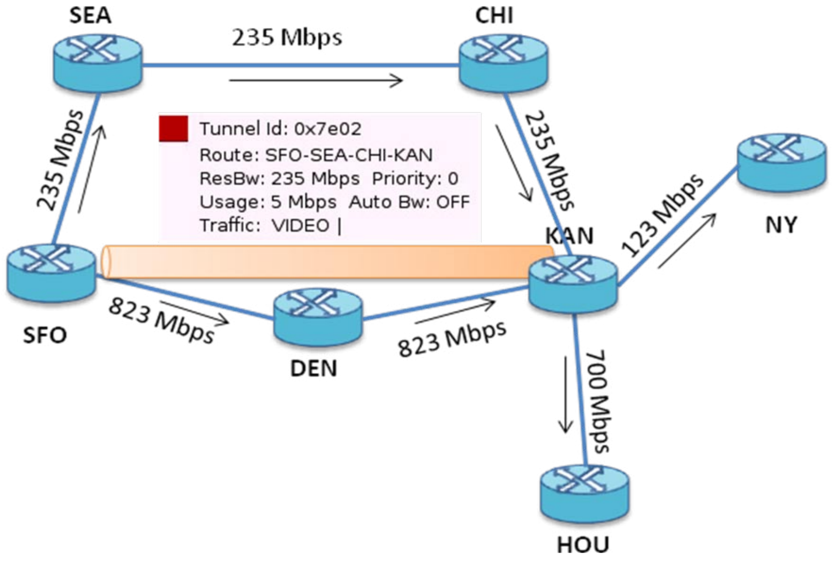
In Fig. 5.14 we show two tunnels that were configured by the network-operator, routed by the CSPF module, and established in the data-path using OpenFlow, to enter flow-entries in the OVS flow-tables. The tunnels originate in SFO (head-end) and terminate in Houston and NYC (tail-ends).

The panels show information for each tunnel. It is worth noting here that Fig. 5.14 shows the tunnels from a conceptual standpoint (similar to Fig. 5.7a). But the tunnels are actually routed over the IP links. For example, the SFO→NYC tunnel is routed along the SFO→DEN→KAN→NYC links, reserving 123 Mbps of bandwidth on each link in the eastward direction. Similarly, the SFO→HOU tunnel is routed via SFO→DEN→KAN→HOU links reserving 700Mbps of bandwidth along the path. But another way to think about this is that this is also the annotated-topology-view presented to the packet-flow routing applications in Fig. 5.9 or Fig. 5.12. The tunnels are, from a packet-flow routing standpoint, uni-directional links which form the shortest (1-hop) path between routers. And so the routing-modules perform Auto-Route, as described in the previous section, by re-routing packet-flows into the tunnels.

For example, flows from SFO→NYC which previously traveled over the IP links now get re-routed (or Auto-Routed) via the 1-hop tunnel. It so happens that the tunnel is also routed along those same links. But this does not matter – the packets get imposed with labels in SFO and get label-switched in the DEN and KAN routers, which no longer have visibility into the IP packet. The flow-entries in DEN and KAN which previously matched on the IP-information in the packet-header, subsequently idle timeout, as they no longer match on the labeled-packets. We also use penultimate hop popping (PHP) of the label at the KAN LSR so that the NYC LSR receives the unlabeled IP packet; and so it only has to do a single lookup to forward the packet.

Another aspect of our design worth pointing out is that for the packet-routing applications to treat the tunnels as unidirectional links, they need to have the concept of a virtual-port. For example, the SFO→NYC tunnel is represented as a unidirectional link connecting a virtual-port of the SFO router to a virtual-port on the NYC router. But the packet-flow routing modules are un-aware that the ports are virtual-ports. They simply decide to forward traffic out of these ports (eg. in SFO) and match on traffic incoming from these ports (eg. in NYC). But these virtual ports *do-not-exist* in the switches, as an OpenFlow switch treats LSPs like any other flow, instead of treating it as virtual ports (tunnel-interfaces). And so the TE- applications translate decisions made by the packet-routing applications from a virtual-port to the corresponding physical-port and label.

<u>Admission-Control</u>: We mentioned that tunnels SFO→NYC and SFO→HOU reserved 123 Mbps and 700 Mbps of bandwidth on all links that are part of their route. Note that in Fig. 5.14 we show these reservations circled in the eastward direction – for example, the KAN→NYC link shows a reservation of 123 Mbps (due to the SFo→NYC tunnel) and the DEN→KAN link shows a reservation of 823 Mbps (due to both tunnels being routed over the SFO→DEN→KAN links).



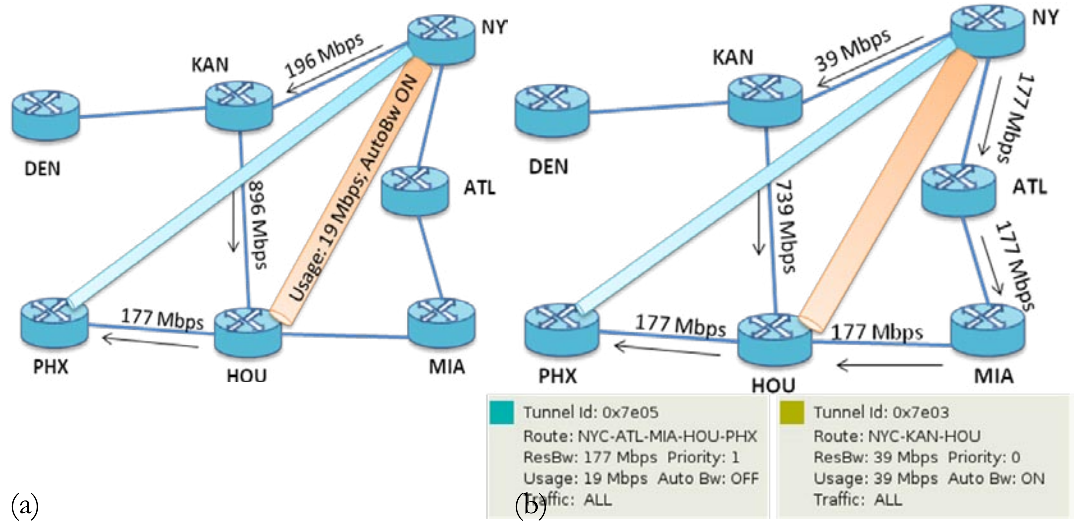
🟥 Tunnel Id: 0x7e02
Route: SFO-SEA-CHI-KAN
ResBw: 235 Mbps Priority: 0
Usage: 5 Mbps Auto Bw: OFF
Traffic: VIDEO |

Fig. 5.15 MPLS-TE Admission-Control

Also note that the maximum reservable bandwidth on any link is 900 Mbps. Thus only 77 Mbps of bandwidth is un-reserved on the SFO→DEN→KAN links. Thus when the SFO→KAN *tunnel* highlighted in Fig. 5.15, requests 235 Mbps of bandwidth, it exceeds the unreserved bandwidth on the shortest path; and so the SFO→KAN tunnel is forced to route along a less-utilized path of the network – the SFO→SEA→CHI→KAN links. This is an example of admission-control. Without it, the SFO→KAN tunnel would be routed along the same SFO→DEN→KAN links as the other two tunnels; and if the tunnels actually used the all the bandwidth they reserved (123+700+235 = 1058 Mbps) it would have congested the links.

Auto-Bandwidth and Priorities: Our next experiment involved the TE features of Auto-Bandwidth and tunnel-priorities and their interaction. Fig. 5.16 highlights two tunnels that originate from NYC: one with a tail-end in Houston (HOU) and the other with a tail-end in Phoenix (PHX).

Note from the panel for the NYC→HOU tunnel, Auto-Bw is turned on (unlike any of the other tunnels in Figs.5.14-15). Accordingly the bandwidth reservation for this tunnel tracks the actual usage of the tunnel. Although the tunnel was created (configured) with a bandwidth-reservation of 10 Mbps, the reservation increases as we channel more traffic through the tunnel (by turning on more traffic-generators); first up to 19 Mbps and then to 39 Mbps (the latter shown in Fig. 5.16).



🟦 Tunnel Id: 0x7e05
Route: NYC-ATL-MIA-HOU-PHX
ResBw: 177 Mbps Priority: 1
Usage: 19 Mbps Auto Bw: OFF
Traffic: ALL

🟨 Tunnel Id: 0x7e03
Route: NYC-KAN-HOU
ResBw: 39 Mbps Priority: 0
Usage: 39 Mbps Auto Bw: ON
Traffic: ALL

Fig. 5.16 Interaction of Auto-Bandwidth and Tunnel Priorities

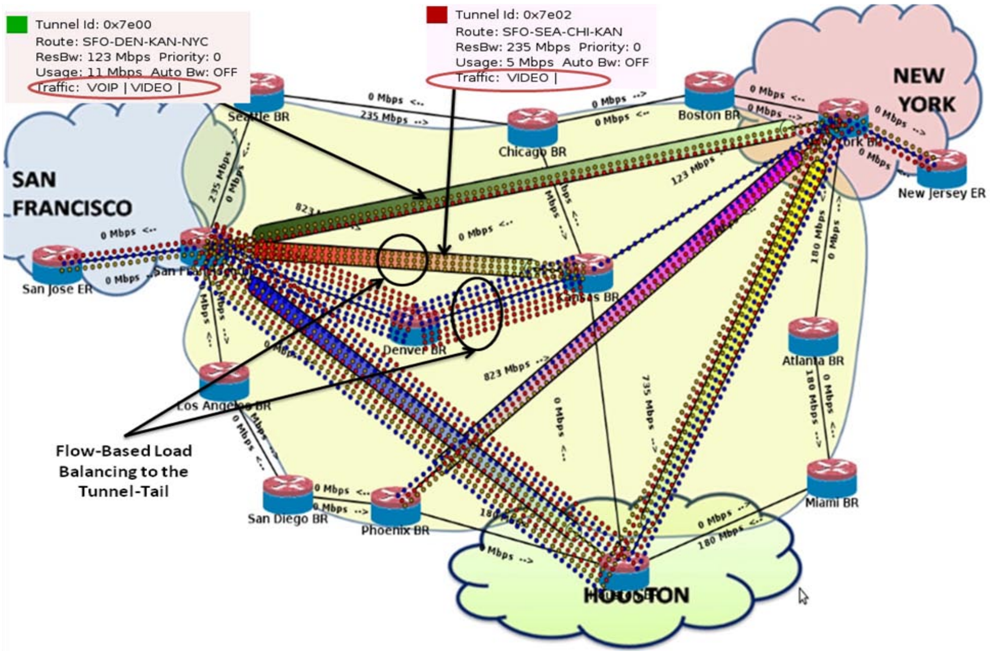
But another effect happens along the way. Note that the route taken by the NYC→HOU tunnel includes the KAN→HOU link and reserves (first 10Mbps and then)

19 Mbps on it. Additionally the SFO→HOU tunnel also reserves 700 Mbps on the same link. The initial route taken by the NYC→PHX tunnel goes via the NYC→KAN →HOU→ PHX links as it is the shortest path (in hop-counts) that satisfies the bandwidth required by the tunnel. Accordingly it too reserves 177 Mbps on the KAN→HOU link (as shown in Fig. 5.16a). This adds up to a total of 896 Mbps reserved on the KAN→HOU link which is very close to the maximum reservable amount of 900 Mbps. And so if we further increase the traffic going through the NYC→HOU tunnel to 39Mbps; then since Auto-Bandwidth is configured ON for this tunnel, it changes the reservation to 39 Mbps (as shown in Fig. 5.16b; which in-turn tries to increase the total reservation on the KAN→HOU link to 916 Mbps, which is not allowed. At this point some tunnel has to find another path.

Note that the NYC→HOU tunnel has priority-0 (highest priority) while the NYC→PHX tunnel has priority-1. Thus it is the lower priority tunnel that is forced to re-route and find another path that can meet its constraints. In Fig. 5.16b, we show that the tunnel has re-routed via the longer NYC→ATL→MIA→HOU→PHX path. This is an example of the dynamic interaction between Auto-Bandwidth and tunnel Priorities, which in a larger network with lots of tunnels, can cause significant network churn.

Class-Based Routing and Load-Balancing: Finally in Fig. 5.17 we show two examples of routing where we deviate from shortest-path routing. Note that Fig. 5.17 is a GUI snapshot that shows all five tunnels we have instantiated so far.

In the first example we implement routing based on traffic-classes. As discussed in the section on DS-TE, tunnels can be created for a traffic-class (identified by the DSCP bits in the IP header) by reserving bandwidth from a sub-pool of link-bandwidth resources. This way per-class admission control can be performed for tunnels designated for a class. The second part of DS-TE is to actually route traffic for that class into the specific tunnel, which today uses cumbersome, non-programmatic techniques. With OpenFlow, routing a traffic-class becomes easy and programmatic. In Fig. 5.17, we show the SFO→KAN tunnel designated to carry only video traffic, while the SFO→NYC tunnel carries both video and VoIP traffic. In the SFO router, we simply use OpenFlow to enter rules that match on both destination IP address and the L4 transport-ports to indentify the tunnel and the traffic-type respectively; and then insert the appropriate label (as an 'action' on matching packets) for the tunnel the packets needs to enter. We can also use the IPv4 DSCP bits to match on a traffic-class (if for example the San Jose router marks the bits).



🟩 Tunnel Id: 0x7e00
Route: SFO-DEN-KAN-NYC
ResBw: 123 Mbps Priority: 0
Usage: 11 Mbps Auto Bw: OFF
Traffic: VOIP | VIDEO |

🟥 Tunnel Id: 0x7e02
Route: SFO-SEA-CHI-KAN
ResBw: 235 Mbps Priority: 0
Usage: 5 Mbps Auto Bw: OFF
Traffic: VIDEO |

Fig. 5.17 GUI Snapshot: Routing of Traffic-Classes and Load Sharing

In this experiment, we do not perform per-class admission control as we did not reserve bandwidth from class specific sub-pools of link bandwidth. But this is a simple change to our CSPF algorithm (and nothing else). More importantly we can create multiple sub-pools each for a different traffic-class; to match with tunnels created for multiple traffic-classes (as shown); which is something that MPLS-TE cannot provide today without changing the routing protocol.

The second example of deviation from shortest-path routing shows load-balancing. In Fig. 5.17, traffic between SFO and KAN takes two routes: Video flows go through a tunnel from SFO→KAN, which is actually routed via the Seattle and Chicago routers; and all other flows take the IP links between SFO→DEN→KAN. This is an example of load-sharing to a tunnel-tail, between a tunnel-path and an IP-link-path; a feature that MPLS-TE cannot provide today due to the nature of Auto-Route in MPLS-TE (as discussed in the previous section).

Conclusions: To summarize, we draw the following conclusions from our SDN based MPLS network prototype:

- We achieved the initial goal of verifying our architectural ideas of introducing the map-abstraction in MPLS networks. We showed that we can not only perform the basic operations of creating and maintaining LSPs in an OpenFlow enabled MPLS data-plane; but we can also provide higher-level services such as MPLS-TE by writing applications above a map-abstraction purely in the control-plane.
- We validated the simplicity of our approach when compared to more traditional approaches seen today. Our implementation of the traffic-engineering application together with all the features described in Sec. 5.2.1, took only 4500 lines-of-code. To accomplish the same in today's network would require implementations of OSPF-TE and RSVP-TE (together ~ 80k LOC) and implementations of all the features. At the time of this writing we could not find an open-source implementation of MPLS-TE features; and so we can only estimate, based on experience that each feature could take at a minimum 5-10k lines-of-code, pushing the total LOC well above 100k. It is easy to see that our SDN based implementation is much simpler given the two-orders of magnitude difference in the lines-of-code required. While our implementations is not production-ready, two orders of magnitude difference gives plenty of room to grow; giving us confidence that even production-ready code will be simpler to implement with the map-abstraction.
- Finally, we achieved our second goal of demonstrating that using SDN and the map-abstraction, we can implement the TE features in ways that a) either greatly simplify the feature; or b) provide a feature that MPLS-TE cannot provide. Examples of the

former include (i) our implementation of Auto-Route in a more flexible and programmatic way; and (ii) contention-free LSP establishment; while examples of the latter include (i) load-sharing traffic to a tunnel-tail, on a tunnel and an IGP path; and (ii) performing per-class admission control for more than one class of traffic.

5.4 Other Applications of SDN based MPLS

The map-abstraction brings the benefits of a simpler, more-extensible control plane to IP/MPLS networks, and provides network-applications with a global-view of network state. Together with the flow-abstraction in the MPLS data-plane and a switch-API like OpenFlow, MPLS networks based on SDN could potentially solve problems seen in today’s networks. In our talks with Tier-1 ISPs, we are beginning to understand issues that carriers face in MPLS networks and how the SDN approach could help meet those challenges. Briefly, we cover a few of these issues in this section.

MPLS-VPNs: VPNs in MPLS come in two main flavors – L3VPNs and L2VPNs (there are variations of the latter). While both services use an MPLS backbone, LSPs in the former carry IP packets, and in the latter, Ethernet frames. Nevertheless, in either case, the primary objective of the MPLS backbone is to provide routing/switching information dissemination and isolation.

Building on our TE work, we recently demonstrated MPLS L3-VPNs [109]. We interactively created multiple isolated L3-VPNs, with overlapping client private-IP address spaces (customer-routes). The controller maintains VRFs for each VPN that a Provider-Edge router is part of. The VPNs also have customer specified topologies, and are supported by TE tunnels in the backbone.

We are beginning to understand the scaling implications of L3-VPNs. Today one of the main problems in L3-VPNs is that the service provider has to carry/propagate millions of customer IP addresses using protocols like MP-BGP. The SDN approach can provide relief in this regard as it can reduce routing burden, by eliminating the need for protocols such as LDP, IS-IS and MP-BGP in the backbone.

Eliminating Auto-Bandwidth: In the previous section we explained how Auto-Bandwidth is a local-optimization technique [110], which together with LSP-priorities, can produce significant network churn. But operators have other options – notably the use of 'offline' (not on the routers) modeling tools, that have global-view of the network and can optimally route LSP paths. However the use of offline-tools introduces a different problem. Once the tools calculate LSP paths for tens-of-thousands of LSPs, the LSPs themselves have to be established or re-routed in a live-network. This process involves 'locking' head-end routers one-by-one and feeding it configuration files for LSPs that it needs to originate (via signaling between routers). The compilation of the configuration takes time, and the order in which tunnels are 'brought-up' has to be carefully coordinated. And all of this has to be done in a network carrying live-traffic.

But with SDN, operators have the ability to take an 'offline' optimization tool 'online'. The map-abstraction offers the full-visibility required by optimization-tools (which can run as applications). But more importantly, the Controller can access the results of the optimization tool and use OpenFlow to directly manipulate the forwarding tables of all LSRs. In other words, OpenFlow acts as a switch-API that *dynamically* updates flow-table state, and therefore LSP state. And it can do so in parallel, for multiple LSPs, originating from multiple head-ends, while minimizing network-churn.

Improving Fast Re-Route Performance: From our talks with network-engineers at Tier-1 ISPs, we find the MPLS Fast Re-Route (FRRs) techniques perform poorly in operation. In FRR, backup LSPs are created to 'locally-repair' primary-LSPs [111]. The backup LSPs are pre-computed and signaled prior to any failure. Local-repair refers to re-directing traffic as close as possible to the failure. Two techniques exist for performing FRR: one-to-one backup and facility backup. But irrespective of the technique used, the problem with FRR is that, in a large ISP network, a) there are far too many nodes and links to backup; and b) because the number of backup tunnels is high, these backup tunnels usually do *not* reserve-bandwidth on the links. And so it is an intractable problem to have any intelligent way for optimally routing backup-paths while backing up everything in the network. Although FRR is fast (< 50ms recovery), it is effectively a blunt protection mechanism. It is entirely possible that the result of an FRR re-route can cause congestion over a backup-path.

The key with SDN and OpenFlow is that it is dynamic - so when a failure happens, one can use FRR as it is today to do what it does today (as meta-stable state), just to give the Controller enough time to optimize around the specific failure. Note that this optimization is no longer an intractable problem, as the Controller knows exactly what failed. And so it does not have to plan for every possible failure, and then it can use OpenFlow to update the LSP-state dynamically once it has optimized around the failure. The net effect is that FRR can perform better by maintaining the speed that comes from local-protection, but availing of optimally routed backup paths that are dynamically updated in LSR flow-table state.

MPLS-TP/Multi-layer Control: Finally we consider applications of MPLS beyond the IP network. MPLS has been favored for establishing a *packet-based data-plane* technology for the transport network; Hence the name MPLS-*Transport Profile* (MPLS-TP) [113]. The basic idea is that there is a need for transport networks that are packet-switched[†], but such packet-networks should retain/reproduce the capabilities of traditional circuit-based transport networking, especially in providing equivalent OAM capabilities. OAM is provided using Generic Associated Channel packets in MPLS LSPs, or pseudowires [114]. Importantly, [113] points out that MPLS-TP should *not* depend on IP networks nor require dynamic control-planes for operation.

However, in our talks with industry, we find that a simple, dynamic control plane like an SDN based control-plane can offer value to an MPLS-TP network. Transport network operators, traditionally against the use of complex control-planes like those in IP/MPLS networks[*], may find simpler OpenFlow based control more to their liking. And its value increases when considered in multi-layer scenarios. Transport network operators

[†] The Related Works section in Ch. 6 discusses MPLS-TP in more detail.

[*] See Challenge #2 in Sec. 3.4

frequently run multiple networks based on different technology layers simultaneously– ROADMs (Layer 0), SONET/SDH (Layer 1), OTN (Layer 1), to which they seek to add MPLS-TP (Layer2). And so instead of doing everything manually, or requiring separate control planes for each layer, a single OpenFlow/SDN based control plane for all layers can be an attractive option.

We believe that we have only started to scratch the surface for SDN applications in MPLS networks. In the next section, we show how we implement one such application.

5.5 Synergies with Packet-Circuit Control

In this section we take a brief look at the synergies between SDN based control of IP networks with MPLS in the data-plane and control of IP networks with Dynamic Circuit Switching. In Fig. 5.18 we show the use of the map-abstraction for control in both cases.

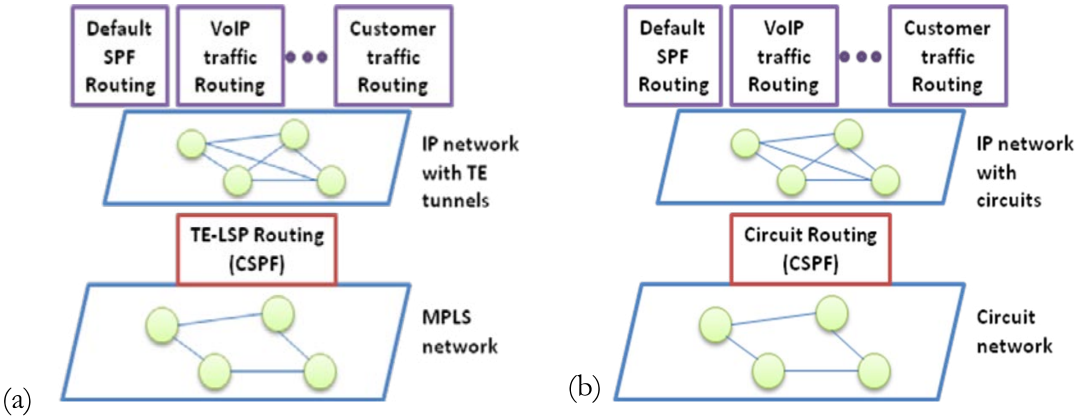


Fig. 5.18 Control of (a) IP and MPLS (b) IP and Dynamic Circuits

In an IP/MPLS network (Fig. 5.18a), the map-abstraction presents an annotated-topology map of the IP network, on top of which we perform TE-LSP routing based on TE-link attributes (like reservable-bandwidth, weight etc.) that we assign to the IP links. The routing uses a CSPF algorithm to determine tunnel routes. When IP is used with dynamic-circuits, in one construction (Fig. 5.18b), the map-abstraction presents an annotated fiber topology, on which circuits are routed using CSPF and attributes we assign to the fiber/WDM-link. The net result of using CSPF in both cases is an MPLS-tunnel or a TDM/wavelength-circuit.

But in both cases they are represented as unidirectional or bidirectional links in the IP network, which is just another annotated-topology map in the SDN controller (the upper ones in Fig. 5.18). In other words circuits and TE-tunnels are represented in similar ways, and both can be dynamic with varying bandwidths and routes. And mapping packet-flows to circuit-flows is really the same as mapping packet-flows to TE tunnels, both of which we have demonstrated in similar ways with voice/video/web packet-flows in Chapters 3 and 5. In both cases, the mapping in the control-plane happens via a virtual-port abstraction[†]. The only difference is that the virtual-port is a purely control plane construct in the MPLS case; while in the circuit-case it represents the data-plane translation from packet-frames (like Ethernet) to framing used in circuit networks.

It is important to note that neither scenario shown in Fig. 5.18 is used today. TE tunnels are not represented as links in the IP network; the IP network does not treat packets as flows; and dynamic-circuits are not used in concert with IP network to dynamically create/modify/delete IP links. Yet the use of the map-abstraction together with the treatment of packets, tunnels and circuits as flows, enables a network-operator or service-provider to have a simple consistent view of their network.

5.6 Summary

The goal of this chapter was to introduce the map-abstraction in MPLS networks, argue for its benefits, and validate our architectural constructs and claims. Our motivation for doing so comes from discussions with service-providers who run large IP/MPLS backbones. In general they feel that while MPLS has delivered on new services like TE and VPNs, it has failed to live up to its promise of a simpler, cost-effective data-plane,

[†] See the sub-section titled 'Tunnel Routing and Packet-Flow Routing' in Section 5.2.2; and the sub-section titled 'Mapping Packet-flows to Circuit-flows' in Sec. 2.1.2.

due in large part to the burden of bearing a tightly-intertwined distributed control plane of ever-increasing complexity.

We showed that the MPLS data-plane is simple and useful in its label-stacking and push/swap/pop operations. And while it is less-flexible than the full flow-abstraction that is inherent in the definition of the OpenFlow protocol, the MPLS use of FECs+LSPs qualifies as 'flows'. But it is the absence of the map-abstraction in the IP/MPLS control plane that makes it complex. So much so that in most cases the only flow-definition (or FEC) actually used is the 'destination- IP address'.

We discussed the benefits of the map-abstraction in the context of MPLS networks. The benefits include simplifying the IP/MPLS control-plane which in-turn reduces protocol load in router CPUs (a real concern in carrier networks where not all hardware is upgraded at the same time). Another benefit of the map-abstraction is that it makes the control-plane more extensible, by not only providing the services that MPLS provides today (TE/VPNs); but also by making it easier to change or introduce new functionality into the network – a fact that can lead to its use as a control plane for MPLS-TP and multi-layer networks. And finally, a key-benefit of the map-abstraction is that it presents a global-view to network-applications which can then globally optimize the network-function or service (as opposed to the local-optimizations possible today). Such global optimizations can help reduce churn (due to Auto-bandwidth) or help MPLS based recovery (FRR) function more efficiently.

We built a network-prototype to verify our architectural constructs and validate our simplicity and extensibility claims. The data-plane switches used the standard MPLS data-plane mechanisms, and the control-plane was based on OpenFlow; on map-abstractions created by a network OS; and network-applications we implemented to replace the functionality of the distributed IP/MPLS control plane. We implemented nearly all the features of MPLS traffic-engineering including TE-LSP admission control, bandwidth reservations, tunnel-priorities, Auto-Route, Auto-Bandwidth, class-based TE, and load-sharing. And we did so in less than 5k lines-of-code, which is nearly two orders of magnitude fewer than implementations using a traditional distributed approach (thereby validating our simplicity claim). Furthermore, we showed that implementing TE features with our design choices can either greatly simplify the feature or provide a capability that the current MPLS-TE control plane cannot provide (thereby validating our extensibility claim). Finally, we discussed how introducing the map-abstraction in MPLS networks fits well with the discussion in the previous chapters on packet-circuit unified network control, in that treating packets, tunnels and circuits as flows, enables a network-operator to benefit from a simple consistent view of their network.

Chapter 6

Conclusions

This thesis described a new approach towards common control, design and operation of converged packet and circuit networks. In WANs today, packet-switched IP networks and circuit-switched Transport networks are separate networks; typically planned, designed and operated by separate divisions, even within the same organization.

Such structure has many shortcomings. First and foremost, it serves to increase the Total Cost of Operations for a carrier – operating two networks separately results in functionality and resource duplication across layers; increased management overhead; and time/labor intensive manual coordination between different teams; all of which contribute towards higher Capex and Opex. Second, such structure means that IP networks today are completely based on packet switching. Which in turn results in a dependence on expensive, power-hungry and sometimes fragile backbone routers; together with massively over-provisioned links; neither of which appear to be scalable in the long run. The Internet core today simply cannot benefit from more scalable circuit *switches*, nor take advantage of dynamic circuit *switching*†. Finally, lack of interaction with the IP network, means that the transport network has no visibility into IP traffic patterns and application requirements. Without interaction with a higher layer, there is often *no need* to support dynamic services, and therefore little use for an automated control plane. As a result the Transport network provider today is essentially a seller of

† As we show in Appendix A, circuit-switches cost less than a 1/10th the price, and can consume less than 1/10h the power and volume of an equivalent packet-switch of the same capacity. Furthermore, there are certain functions *dynamic* circuits perform exceedingly well in the core – recovery, BoD, guarantees – that IP networks cannot take advantage of today.

dumb-pipes that remain largely static and under the provider's manual control, where bringing up a new circuit to support a service can take weeks or months.

Our proposal addresses these issues by offering a new unified-control-architecture for carrier networks. Our solution was designed around two control-abstractions based on an emerging idea called Software Defined Networking (SDN). The first abstraction we proposed is a common-flow abstraction that fits well with both types of networks and provides a common paradigm for control. It is based on a data-abstraction of switch flow-tables, manipulated by a common switch-API. The flow-tables take the form of lookup-tables in packet switches and cross-connect tables in circuit switches. Together with the switch-API, it abstracts away layer and vendor specific hardware and interfaces, while providing a flexible forwarding plane for manipulation by a common control plane. The second abstraction we proposed is a common-map abstraction, based on a data-abstraction of a centralized network-wide common-map, manipulated by a network-API. The common-map has full visibility into both packet and circuit switches networks, allowing creation of network applications that work across packets and circuits. Full visibility allows applications to jointly and globally optimize network-functions and services across multiple layers. And implementing network-functions as centralized-applications is simple and extensible, as the common-map abstraction hides the details of state-distribution from the applications.

We implemented these guiding principles in three network-prototypes we built to help us progressively verify our architectural constructs and ultimately validate our architectural claims. Two early prototypes focused on the common-flow abstraction for packet-switches working in concert with different kinds of circuit switches. The third, more complete prototype helped us understand the intricacies of building a converged network with a common-map and network-API. With this prototype we built a testbed that emulated WAN structure; and then implemented a new and fairly involved network capability which benefits from both packet and circuit switching.

With this new network capability implemented on our prototype, we validated our simplicity and extensibility claims, by comparing our implementation to our-best-guess on what it would take to implement the same functions using industry-standard solutions for packet-circuit control. We found our solution to be two-orders-of-magnitude simpler (in terms of line-of-code) compared to the industry solution, thereby validating our simplicity argument. More importantly, we presented qualitative architectural insights into why our solution is more extensible than current industry-solutions. We believe there are two main reasons why current packet-circuit control solutions are hard to use and extend: a) the use of the UNI interface which results in loss of visibility at the intersection of packet and circuit networks; and b) the implementation of services/network-functions as distributed systems which are tied to the state-distribution mechanisms. We argued that since our solution does not suffer from these limitations, it makes it easy to introduce new functionality into the network, or change existing functionality just as easily; thereby verifying our extensibility claim.

This latter fact bodes well for service providers who find it hard to differentiate their service-offerings from other carriers. Their networks today are built using closed-systems (routers and switches) from the same set of vendors with the same set of features. But with the use of our control-architecture, carriers can innovate in their networks across packets and circuits, and find ways to provide new revenue generating services without having to implement the features and services in the routers themselves. Put simply, our control architecture can lead to a faster pace of innovation and service differentiation.

At the same time carriers that own both kinds of networks, can benefit from reduced TCO by operating one converged network instead of two separate ones. Our control solution already helps converge the operation of the different switching technologies, while allowing the network-operator the flexibility to choose the right-mix of technologies for the services they provide. It also paves the path for developing common management tools, planning and designing network upgrades by a single team, and reducing functionality and resource duplication across layers.

As an example of the potential savings possible in a converged packet-circuit network with our unified control architecture, we performed a Capex and Opex analysis on a nationwide US backbone IP network. We considered industry standard practices for IP network design as reference design (IP-over-WDM). We then proposed a converged network we call IP-and-DCS, based of the following: a) replacing all backbone routers in PoPs with packet-optical switches; b) using a full-mesh of variable bandwidth circuits between PoPs; and c) adopting our unified control architecture for common control of packet and circuit switching. The latter feature is especially important in realizing the full-mesh topology, and for enabling close interaction between packets, dynamic circuits, and application/service needs. Interestingly we found that such construction can save us as much as 60% in Capex costs and nearly 40% in Opex costs when compared to the reference design. Importantly, such savings are insensitive to varying traffic matrices; and scale better (at a lower $/Tbps slope) as the network is upgraded to accommodate 5X or greater traffic-growth. We expect even greater Opex savings, as the use of a unified control plane eliminates time and labor intensive manual coordination and provisioning between IP and transport teams.

Even if carriers prefer a less integrated approach than the one described above, IP networks can still benefit from dynamic circuit-switching in transport networks, by leveraging a key aspect of our unified-control architecture: *slicing*. A network-slice according to our definition is a combination of bandwidth resources and control over switching resources. With slicing, a transport service provider can carve up slices of their networks and sell them as a new service to ISP. It gives transport-SPs incentive to share information with ISPs which is critical towards the creation of the common-map. ISPs can then create common-maps of their packet network and their slice of the transport network, to jointly optimize services across them and enable network-applications such as those we have described: dynamic-links, variable-bandwidth links, application-aware traffic-engineering, unified-recovery and others. Furthermore offering slices presents a new revenue opportunity for transport service providers, making them more than dumb-pipe sellers – a crucial fact given that all services are moving to IP. Importantly slicing also helps the gradual adoption of our unified control architecture into today's networks. Without gradual adoption the successful implementation of any new-control technology would only be limited to green-field deployments, which are a rarity in core networks.

We have also applied the SDN based control architecture to MPLS networks. While MPLS has delivered on services (TE, VPNs), operators feel that it has failed to deliver on the promise of cheaper, inexpensive switching; due in large part to the burden of bearing a tightly coupled, distributed IP/MPLS control plane of ever-increasing complexity. We showed that by retaining the MPLS *data*-plane (flow-abstraction) and introducing a simpler and extensible control-plane (map-abstraction), we can *provide the services* that MPLS provides, *without the complexity* of the IP/MPLS control plane. The implications are many-fold – simpler switching hardware based solely on MPLS (and supporting just OpenFlow) can be realized; all existing MPLS services can be supported, made simpler, more-dynamic and optimized; or new services/functionality can be created on MPLS rapidly without requiring extensions to existing protocols. To demonstrate our approach, we showed nearly every major feature of MPLS-TE that we implemented in just a few thousand lines of code; compared to nearly two orders-of-magnitude greater code-lines required in traditional distributed approaches.

Our work is in the early stages. Many challenges remain, the most important of which is the development of the common-map-abstraction by a network-operating system that is designed to operate on multiple-servers in multiple geographic locations, to ease scalability and performance concerns. Work is underway in industry and academia to realize such a control plane and meet the related challenges.

We are convinced that (with further development) if the ideas in this thesis are adopted by service providers, the greatest impact would be that they can remain profitable as the Internet grows. As a result, they would have greater incentive to invest in their networks, which in-turn would benefit society immensely.

6.1 Related Work

Our unified control architecture for packet and circuit networks is based on Software Defined Networking (SDN) principles [17]. There is a rich history of related ideas both in academia and industry. The 4D project [138], RCP [139] and the SANE and ETHANE [18] works are part of a related line of research that advocate the separation of control and data planes in packet networks.

In the industry, Ipsilon's General Switch Management Protocol (GSMP [140]) also allowed an external controller to control one or more ATM switches. GSMP was later extended to control other kinds of label-switches such as MPLS and Frame-Relay switches [141]. GSMP was not used widely; mainly because ATM and Frame Relay networks fell out of favor, and dominant vendors chose to implement the IP/MPLS control plane for packet-networks.

The IETF ForCES framework [142] defines a set of standard mechanisms and guidelines for control and forwarding separation, which is similar in spirit to OpenFlow; but it does not define a specific protocol like OpenFlow does to control a forwarding plane abstraction. The ForCES framework does not advocate the SDN map-abstraction or the flow-abstraction. It only seeks to define a framework for standardizing the exchange of information between third-party router-control plane software and off-the-shelf forwarding hardware.

We are the first to propose the use of SDN ideas to develop a unified control architecture for packets *and* circuits. None of the works mentioned above are related to circuit networks. The only prior work for unified packet and circuit control is MPLS/GMPLS. And so we take a closer look at MPLS/GMPLS in the Section 6.1.1 and contrast it with our SDN based ideas for packet-circuit control. We also take a closer look at a related data-plane technology called MPLS-TP that is being introduced in transport networks. In Sec. 6.1.2 we discuss IP and MPLS-TP networks in keeping with the original premise of this thesis – finding a way to run *one* network instead of two.

6.1.1 MPLS/GMPLS and ASON

Generalized Multi-Protocol Label Switching (GMPLS) was designed as a superset of MPLS, and intended to offer an intelligent and automated *unified control plane* (UCP) for a variety of networking technologies – both packet and circuit (see Fig. 1 from [24]).

Conceptually, Generalized Multi-protocol Label Switching (GMPLS) in the *data-plane*, uses the core idea of treating packets as flows/LSPs, and extends it to the circuit-domain where circuits were recognized as LSPs as well† [58]. Thus in the data-plane both packets and circuits could be treated as LSPs (similar to our common-flow abstraction). But in the *control-plane*, GMPLS ended up adopting the MPLS control plane protocols (OSPF-TE, RSVP-TE) as well. Since MPLS already had a well developed control plane (derived from a well-developed IP control plane), GMPLS simply extended the same distributed routing and signaling protocols (OSPF-TE, RSVP-TE) to control circuit switches [20, 23-25]. The reasoning given for adopting the MPLS control-protocols was to “avoid re-inventing the wheel”, by using existing protocols and merely extending them for the unique characteristics of the circuit-world. It was thought that the use of the same protocols could lead to an automated, unified control plane for a variety of technologies – packet, time-slots, wavelengths, and fibers.

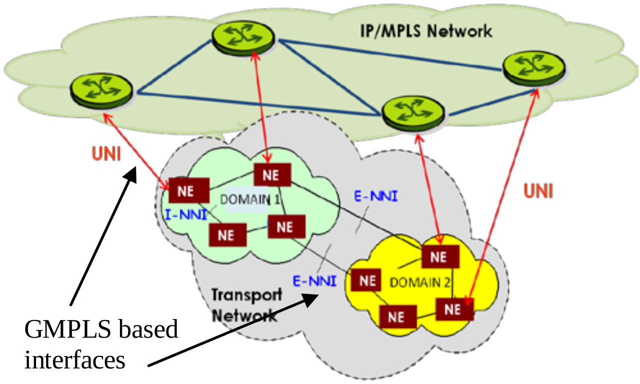
But at the end of the day, GMPLS was still just a collection of control protocols. There was a need to fit GMPLS to an architectural-model (better understood as a usage model) for the interaction between IP and Transport networks. The IETF defined three such models – peer, overlay and augmented. They differed by the amount of state that was shared between the IP and transport network: all, none and some, respectively [19].

In principle you could use a *single* instance of GMPLS protocols across packet and circuit domains, treating it as a unified-control-plane with full information sharing between routers and transport switches, as proposed by the IETF *peer* model [19]. But the peer model has never found favor for mostly two reasons: it ignores organizational boundaries where information sharing is prohibited; and it increases load on (fragile)

† with ‘implicit’ labels – like the color of a wavelength

routing protocols, which now have to distribute not just packet-switch and link information, but transport-switch and link information as well.

What did find favor (at least in other standards bodies) is the IETF *overlay* model [19], which was formalized by the ITU into its own architectural description for Automatically Switched Optical Networks (ASON) [21]. It took into account existing transport architecture and organizational boundaries. The overlay model separates the control of IP and transport networks, by treating the IP network as an overlay network, which shares no information with the transport network and crucially, runs a completely separate instance of the control plane – i.e. the IP/MPLS network runs the IP/MPLS control plane and the transport network runs the GMPLS control plane, and no information about the networks is shared across any boundary/interface between the networks in either direction. Such an interface between the IP and transport network is known as a User-Network-Interface (UNI) [†].



UNI: User-Network Interface between client and transport network

E-NNI: Exterior-Network Network Interface between vendor-islands

I-NNI: Interior-Network Network Interface within

Figure 6.1: ITU ASON model [21], similar to GMPLS overlay-model [19]

But the ITU went further. It not only required the UNI, but also defined other interfaces that maintained existing transport network structure – ie. vendor-islands and proprietary interfaces (Fig. 6.1). The transport switches in the vendor-islands would continue to be controlled by vendor-proprietary solutions, but those solutions would be made to inter-work by *layering GMPLS protocols on top* of those solutions, via interfaces knows an E-NNI. The IP network can make a request for services from the transport

[†] The IP network is the User and the transport network is the Network. The OIF UNI is an example of this interface [22]. Not surprisingly IETF has proposed its own UNI as well (RFC 4208).

network using what the UNI provides. Such a request is then relayed (and satisfied) across vendor domains in the transport network via the E-NNI interface.

We make a few observations here on MPLS and GMPLS. While MPLS has been very successful in wide-area IP networks, GMPLS has been a failure in terms of actual deployment. It has undergone a lengthy standardization process at the ITU, IETF and OIF; all the transport equipment vendors have implemented it in their network elements; there have been many interoperability demonstrations by vendors; and yet after a decade, there hasn't been even *one* significant commercial deployment of GMPLS as a *unified* control plane.

GMPLS may have found use in a limited way as a tool to help the NMS/EMS provision a circuit after being triggered manually by the management system. But we are not interested in such use of GMPLS. Such use has a) nothing to do with the IP network; and b) nothing to do even with the ASON view of using GMPLS as an interoperable *transport* control plane. Rather this use of GMPLS is merely a proprietary vendor implementation of a control plane just for their switching nodes (in other words as a vendor-island I-NNI).

We are *not interested* in creating a control plane for Transport networks. On their own, transport networks *do not need* dynamic automated control, as they do not provide services that require dynamic circuit switching. We have discussed this in the Introduction, but it worth repeating – all services are moving to IP; and it is only the IP network that supports services diverse enough to benefit from dynamic circuit switching. And so we are solely focused on creating a simple unified control architecture across *both* packets and circuits, where the benefits of both switching technologies can be taken advantage of from a common network-application standpoint.

It is in this context that GMPLS is an utter failure. We do believe that the initial choice to use the concept of a flow in the data plane as the common abstraction was the right one. However, the subsequent choices made or overlooked have contributed to its failure. We offer our perspective on why GMPLS fails as a UCP:

- Control Plane Complexity: Using the MPLS control plane as the starting point, results in protocols overly complex and fragile to make sense as a UCP. For example:
    - Dynamic Link-State Routing Protocols: Distributed link-state routing protocols like OSPF or IS-IS have convergence and stability issues if network state changes too fast or too often. This is the fundamental reason why IP networks today do not support dynamic links or dynamic link weights. To extend OSPF and use it in a dynamic circuit network with its effect being felt by the same or another instance of OSPF in the packet network is dangerous and unwarranted. We discussed this in more detail in Sec. 3.4 (Challenge #3).
    - Protocol Re-Use: Furthermore, the reasoning given for starting with the MPLS suite of protocols to was to "avoid re-inventing the wheel", by using existing protocols and merely extending them. However, with the amount of extension that has gone into the protocols, this is simply not true anymore. Consider RSVP – it was originally intended for hosts to signal for resources from networks in the IntServ architecture; but then extended to serve as an LSP signaling mechanism in MPLS-TE; extended again to include signaling for transport network LSPs; and modified a third time to support the UNI interface. Every extension carried baggage from the previous extension increasing code-bloat and complexity.
    - Inflexible network architecture: To further increase the complexity of the control plane, GMPLS in the overlay model (Fig. 6.2) insists on retaining transport network vendor-islands; layered architecture *within* the transport network; and new interfaces (UNI/E-NNI) to glue them all together. While retaining vendor islands and proprietary interfaces (and adding GMPLS on top) may help with backward compatibility to deployed hardware; in reality it makes the overall solution so complex that no one tries it in production.

In our unified control architecture, we eliminate protocols like OSPF and RSVP, interfaces like the UNI, vendor-islands and vendor-proprietary interfaces. We replace them with a common switch-API (OpenFlow), a slicing-plane, a common-map and a network-API. In the process we drastically reduce the control-plane complexity, as evidenced by our analysis in Sec. 3.2.

- Lack of the common map-abstraction: In Chapter 3, we pointed out that the common-map abstraction and network-API (supported by the slicing plane) gives full visibility across packets and circuits, and isolates the implementation of network-functions from the state distribution mechanisms (Sec. 3.3.1). We argued that this is the right abstraction: as full-visibility allows joint and global optimization of services and networking functions across packets and circuits; and abstracting away the state-dissemination mechanisms results in functions that can be written in a centralized way, making the control plane simple and extensible. GMPLS lacks the map-abstraction – it limits the services available to the IP network to the exact service-level definitions defined (and pre-baked into the infrastructure) by the UNI interface; and distributed implementation of applications across packets and circuits require lots of glue-code, patchwork to existing protocols, or new protocols. In short both features hamper simplicity, extensibility and kill programmability; ultimately limiting its use.
- Lack of a gradual adoption path: Finally GMPLS provides no means for flexible and gradual adoption of a new control plane. Network operators are conservative. Transport network operators would like to respond faster and provide more dynamic services to meet their client needs, but loathe giving up precise manual control over the way traffic is routed over their network to a software control plane, irrespective of how intelligent that control plane may be. Additionally any new control technology has to compete with decades of established operational procedures. So the real key is to offer a way in which a network operator could gradually try out a new technology in a slice of the network to gain confidence in its abilities, while at the same time being able to flexibly choose the correct mix of technologies for a service. In our control architecture we provide a gradual adoption path via incremental deployment using a slicing-plane (see Sec. 3.4, Challenge #2).

Thus we find that GMPLS is completely un-usable when it comes to easing the inter-working of packet and circuit switching. It is too complex; too buttoned-down, too inflexible to be of any use from a common network-function or service standpoint across packets and circuits. IP networks will not touch it. Transport networks find little use for it. And so, it comes as no surprise that it has never been used commercially across packets and circuits.

This is unfortunate, as the benefits of dynamic-circuit-switching to packet networks are real and tangible (as we have shown in Chs. 3 and 4); and should not be shrouded by the deficiencies of the control-mechanism. We hope that the adoption of our unified-control-architecture can help service providers benefit from the advantages of both switching technologies.

6.1.2 MPLS-TP and Packet-Optical Transport Evolution

The last few years have seen an emerging trend in transport networks to move away from older TDM circuit-switched technologies like SONET/SDH; and include in its stead, packet-transport technology like MPLS-Transport Profile (MPLS-TP). MPLS-TP adds OAM functionality, which is missing in regular MPLS. It does so by defining a new reserved label-id, and new payloads such as the generic-associated-channel (G-ACh) which carry OAM information in an LSP [126]. The need to add OAM comes from the fact that while transport SPs wish to move away from SONET, they want to retain the rich OAM capabilities provided by SONET. Unfortunately, they also want to run MPLS-TP networks like they run SONET networks – manually, without a control plane [127].

At the same time some service-providers such as Verizon, believe that packet-transport like MPLS-TP is needed in *addition* to newer circuit technologies like OTN and multi-degree ROADMs [89, 90]. And so many vendors have planned product releases for such Packet-Optical Transport Switches [85-88].

While it is still murky as to which direction future transport systems will go, this much is clear – the idea to build packetized-transport is clearly a reflection of the extremely high costs involved in supporting all services on an IP core-network [128]. Since all services are moving to packet-networks; by building a packet-network *in parallel to the Internet core*, the hope is that some packet-services can be siphoned away from the costly IP network and provided by a packetized-transport network. In this way reduced load in the IP network could result in requiring lesser IP router-ports and thereby reducing Capex.

We believe that the idea of supporting services by introducing cheaper MPLS-TP ports is a sound one. But doing so in a network parallel to the IP network, neglects the fact that *two separate networks* will still need to be operated. Even if both are predominantly packet networks, their operational modes will be quite different, given that the MPLS-TP network is intended to be run with centralized control and supported/ supplemented with OTN and ROADMs. Instead, we have proposed an integrated IP/MPLS(-TP)/OTN network in Chapter 4, which not only achieves the desired Capex benefits, but also the Opex benefits of operating a single network with a multi-layer control plane.

6.2 Future Work

6.2.1 Challenges in the Control Plane

In this section, we discuss the challenges our architectural choices face, and highlight the work being done by others to overcome them.

Scale: A common perception of centralized decision making is that it cannot scale when compared to distributed solutions. However, with the common-map abstraction in the SDN approach, we note that only the control programs are implemented in a centralized way with a global view of the network. The common-map abstraction *itself* is

† At the time of this writing, MPLS-TP seems to be the technology of choice, winning out over older 'Carrier Ethernet' offerings like T-MPLS and PBB-TE.

created by a Network Operating-System, which is meant to be distributed and built to scale.

A question that is often asked regarding scale is: How many flows per second can the Controller handle? This is essentially a question regarding Controller throughput which can be a raw indicator of scale. [143] offers a side-by-side comparison of raw throughput for several open-source controllers. We note that a single server (4 cores, 8 threads) can process several millions of flows/sec from a group of 32 (emulated) switches, irrespective of the type of controller used. However this result needs to be put in perspective:

a) This result is essentially an indicator of the speed of response of the Network-OS software stack, which includes the TCP/IP and OpenFlow message-handling stack and a basic-network application like an Ethernet learning-switch. In general, when it comes to throughput, the controller's compute and memory resources are not limiting performance factors [130]. Trivially it is possible that a controller realized by a cluster of servers can replicate all the compute/memory resources found in routers in a fully distributed solution today. Such a cluster could run the same distributed algorithms as routers do today. But more importantly, such a compute cluster can far outperform a collection of routers in terms of compute and memory resources[†]. However in most cases the actual flows/sec handled will vary widely and depend on i) the network-application; ii) the state-distribution mechanism; and iii) the consistency overhead one must pay for maintaining (reading/writing) network state. However, the authors in [130] argue that such consistency need only be maintained on a per network event timescale (hundreds or thousands/sec), instead of a per-flow (millions/sec) or per-packet (billions/sec) timescale. And so maintaining (eventual) consistency for a (relatively) small number of events per sec is what allows the Controller to scale.

b) Another fact worth noting about the results in [143] is that they are produced from a Controller running on a single server. In our prototypes we used NOX which is also designed to run on a single server. However more advanced (commercial) network-

[†] For example, router/switch CPUs are typically several generations behind server CPUs.

OSes [129] are designed to run on multiple servers to account for performance scale and resiliency, which are key requirements in production networks. In [129] the authors also argue for the use of well-known, general-purpose techniques* from the distributed-systems community, for the problem of state distribution and maintaining consistency and resiliency, in lieu of more specialized distributed routing protocols.

c) In most cases today we find that what limits throughput is not the Controller performance, but the switch CPU performance. The rate at which the switches generate events for the controller (like packet-ins), as well as the rate at which flow-messages from the Controller take effect in the data-plane, are both limiting factors. The primary cause is poor switch-ASIC to CPU bandwidth. We believe that as SDN popularity increases, systems will be built which will optimize for this mode of operation thereby bringing down the timescales involved.

d) The other issue in controller scaling is that of geographical scale. In wide-area networks, not only is it necessary to have the network-OS be supported over multiple servers at one geographic location, but it is likely that controllers will be located in multiple geographic locations, both within an AS and across multiple ASes. Naturally some Controller-to-Controller communication mechanism will be required. While still early days for SDN use in the wide-area, potential solutions have been discussed in [131]. We anticipate this to be a rich area of networking research in the near future.

Latency: Another significant objection to centralized decision making is the round-trip-time (RTT) it takes to go ‘outside-the-box’ to a controller for decisions. While it is true that there is a cost, the actual effect on network performance can vary depending on the way SDN is deployed.

If deployed in a way that every new flow must be routed ‘reactively’ by sending the first packet of the flow to the controller, then the RTT cost must be paid. The RTT is composed of the following times:

1. Time to generate an event (and its corresponding message) in a switch

* Examples are Zookeeper, Dynamo, Cassandra etc.

2. Time to propagate the event message to the controller
3. Time for the controller to act on the event and make a decision
4. Time for the decision to be propagated back to the switch
5. Time for the decision to actually take effect in the switch

Time intervals for #2, 3 and 4 collectively have been measured in the 100's of microseconds for local controllers that perform basic routing of flows [130]. This corresponds to controller throughputs of hundreds of thousands of flows per sec. In the wide-area the propagation times (#2 and #4) would increase, implying the need for placing controllers optimally. And as we have mentioned before, while today's switches exhibit longer times for #1 and #5, we expect that optimized switching systems will reduce this time component. Nevertheless an RTT cost must still be paid for the first packet of a new flow in this mode of operation.

But there is another mode of operation that can keep latencies low. For example, wide-area networks are often pre-planned (both MPLS and circuit) where primary paths for flows have backup paths and secondary backup paths that the switches are already aware of and can switch to for fast-failover. Such 'proactive' mechanisms are possible in SDN as well. The Controller can pre-compute and download primary and backup information in the switches beforehand, so that packets stay in the data plane when failures happen.

Another mechanism involves the use of default paths for all new flows. Such paths are already pre-installed (with lowest priority of match) in the switches chosen to be along the default path (possibly the shortest path). This way, data packets never leave the data-plane. Occasionally the controller samples the packets matching the default flow (by using something like sFlow [144]), and figures out what to do with them – again in this scheme new flows do not suffer the RTT cost [132].

Application Isolation: One of the defining characteristics of the common-flow abstraction is the ability to determine and resolve conflicts between decisions made by different network applications running on the same network OS. The basic idea is to help extensibility by not requiring new network applications to take into account other applications already in play.

Our implementation of the abstraction does not provide this isolation comprehensively. For example, prioritizing (how event-handlers in different applications are called for the same event) helps - one could write an application for a specific case of a more generic event, prioritize its event handler higher than the generic handler and then stop execution of the event after the special handler deals with it. But it does not provide isolation generically. Even with event prioritization, an application writer has to take into account all other applications that have subscribed to that event, which in-turn sometimes requires knowledge of how that application handles the event.

Some early and interesting work that takes a different approach to application-isolation can be found in [133].

OpenFlow: The OpenFlow protocol serves as a switch-interface that instantiates of the common-flow abstraction. However the protocol has some limitations in terms of its structure (c-structs instead of TLVs or protobufs), and in the recent past it did not model real switch hardware well enough. The latest version of the protocol overcomes some of the limitations (multiple-tables in v1.1) and a new industry organization (the Open Networking Foundation) has been formed to further the development of the protocol [134]. The other factor involving hardware abstraction is that OpenFlow requires the ability to perform very flexible packet matching (combination of L2-L4), something that can be performed with TCAMs, but current hardware has small TCAM space. Instead they have large tables that can perform specific matches (just for L2 or just for L3). We expect that as more switches come to market optimized for OpenFlow this may change as well. Similarly our extensions to the protocol for circuit-switching are experimental and not comprehensive – for example, OTN switching is not supported and wavelength switching has only rudimentary support. But with the formation of the ONF, it is expected that the ability to control all kinds of circuit switches will eventually be included in the OpenFlow specification as well.

6.2.2 Applying pac.c to Other Network Domains

Recently there has been discussion of introducing optical switching in non-traditional networking domains such as datacenters [135, 136]. The advantages of optical circuit switching (guaranteed bandwidth, low-latency, creating new packet-links etc.) working in parallel with traditional packet switching has been recognized. Furthermore, in all proposals, OpenFlow and SDN ideas have been used or proposed in the control plane for control of packet and circuit switches [137].

We have similar views. While in this thesis, we have focused on traditional use of circuit switching in WANs, a lot of our ideas and conclusions can be transferred to other networking domains. The references mentioned above largely treat packet and circuit switching as *parallel* networks accessible to Top-of-Rack (ToR) switches. In addition, we believe that optical switching can be *integrated* into a packet-network as well in limited or extensive ways.

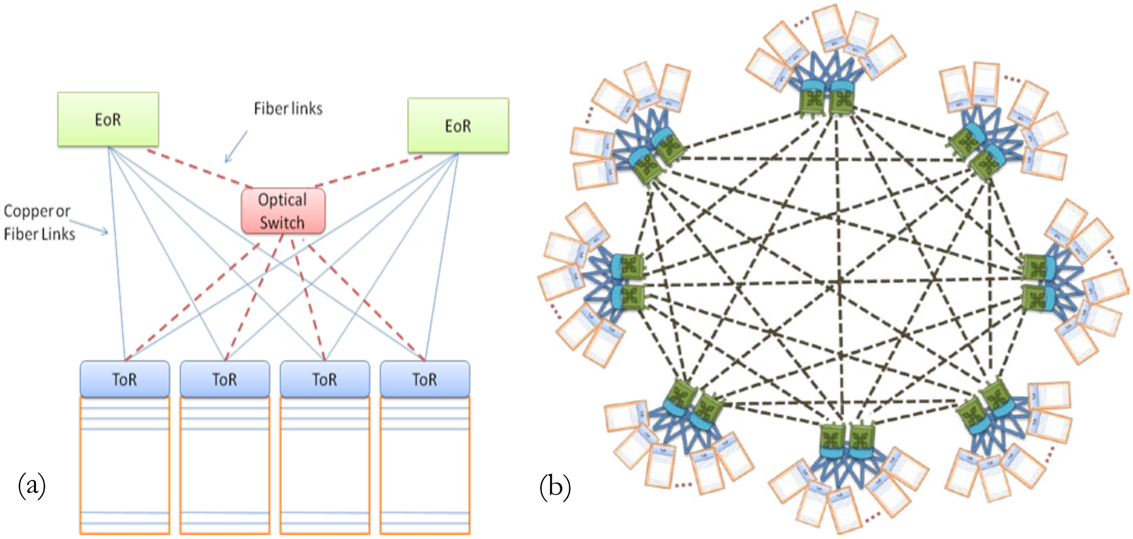


Fig. 6.2 Integrated Approaches for Packet and Circuit Switching in Datacenters

Under one construction (Fig. 6.2a), we can introduce small optical cross-connects *in-between* the levels of packet-switching hierarchy, with the specific purpose of creating *new links* between the switches they interconnect at different levels. These links are created when needed, to augment the bandwidth available between switches and routers. With this construction, we can temporarily change the over-subscription ratio by redirecting bandwidth when needed for as long as it is needed. For example, if the link between ToR1 and EoR1 is congested, we create a new link by cross-connecting their spare interfaces inside the optical fiber-switch. At a later time, if necessary, EoR1s spare interface can be connected to a different ToR switch.

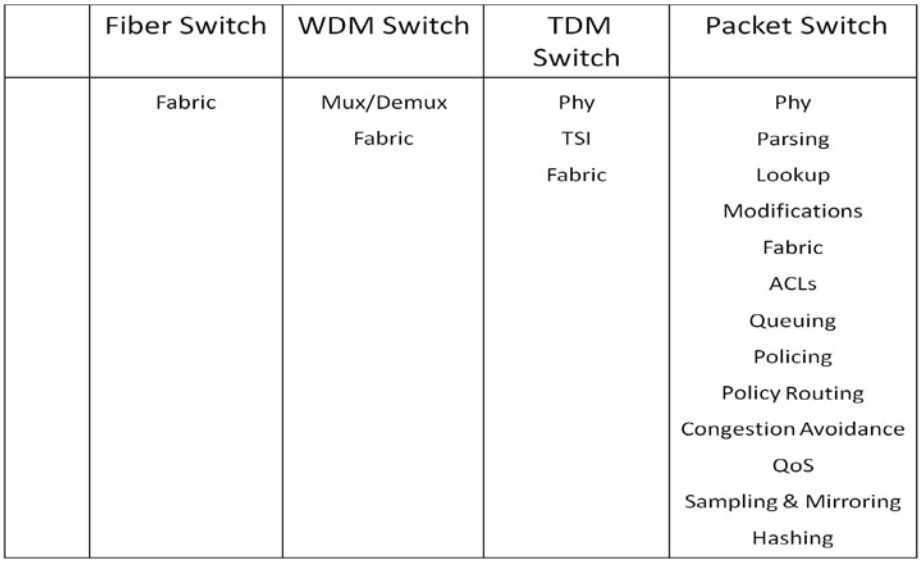
In another construction (Fig. 6.2b), our IP-and-DCS design choices in Chapter 4 could translate to the datacenter as well. For example, a cluster of servers can be created with high bisection-bandwidth and low latency by using one-hop paths between all ToR switches. These ToR switches essentially take the place of the Access Routers in the IP-and-DCS WAN design. The one-hop paths between all ToR switches are supported by a full-mesh of dynamic-circuits between Packet-Optical switches. As before our unified control architecture is adopted to make the full mesh realizable; and to control the dynamic interaction between packets, circuits and services.

We expect that there will continue to be interest in applying packet and circuit switching together in datacenters and possibly even enterprise campus-networks. We believe that such constructions can avail of the benefits of both switching technologies only if used from a common application viewpoint that our control architecture provides.

Appendix A

Packet & Circuit Switches

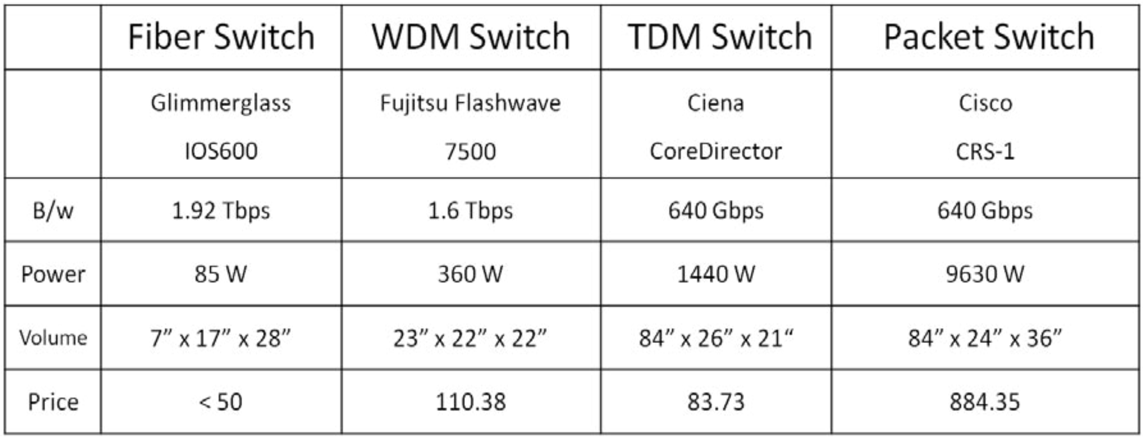
In Chapter 1 we briefly touched on the internal structure of a typical circuit switch (Fig. 1.8), including input /output ports or linecards and a switching fabric. Depending on whether this fabric is digital or optical, there may be "Phy" chips on the linec-ards; and depending on whether it's a wavelength, time-slot or fiber switch, there may be another chip doing time-slot interchange or wavelength mux/demux. On the other hand, there are many more functions that are performed by packet switches. Table A.1 lists most of these functions.



	Fiber Switch	WDM Switch	TDM Switch	Packet Switch
	Fabric	Mux/Demux	Phy	Phy
		Fabric	TSI	Parsing
			Fabric	Lookup
				Modifications
				Fabric
				ACLs
				Queuing
				Policing
				Policy Routing
				Congestion Avoidance
				QoS
				Sampling & Mirroring
				Hashing

Table A.1: Comparison of Packet and Circuit switching functions

Comparing the switches side by side, it is reasonable to expect that since packet switches do much more, and they do it at a much smaller granularity and in much faster time scales than a circuit, it would come at the cost of power, size and price. When we compare real-world values, it matches our expectation quite well. Table A.2 lists four high capacity high-end switches – 3 circuit switches and 1 packet switch. It also lists their power consumption, volume and price. The price numbers are relative numbers derived from [82] - a good rule of thumb would be to multiply the numbers with $1000 to get absolute values.



	Fiber Switch	WDM Switch	TDM Switch	Packet Switch
	Glimmerglass IOS600	Fujitsu Flashwave 7500	Ciena CoreDirector	Cisco CRS-1
B/w	1.92 Tbps	1.6 Tbps	640 Gbps	640 Gbps
Power	85 W	360 W	1440 W	9630 W
Volume	7” x 17” x 28”	23” x 22” x 22”	84” x 26” x 21“	84” x 24” x 36”
Price	< 50	110.38	83.73	884.35

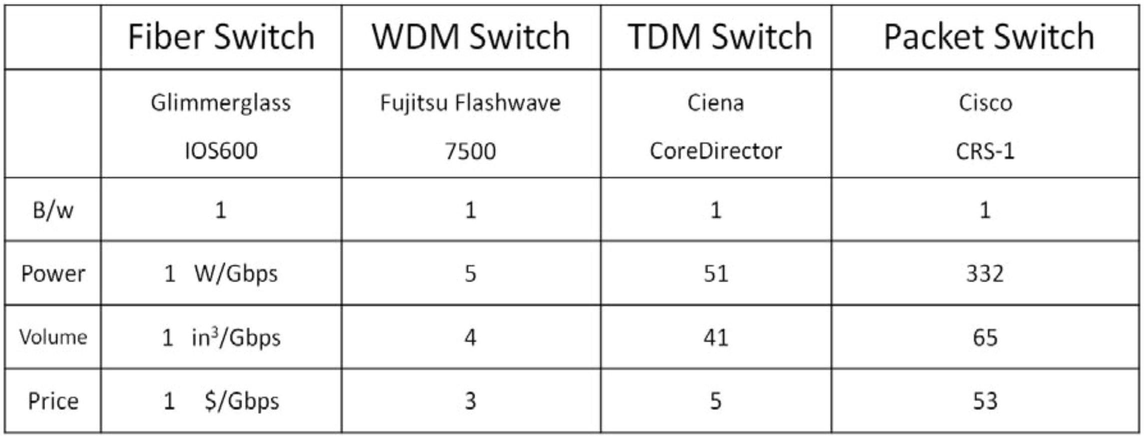
Table A.2: Power consumption, Size and Price Comparison

Price Calculations:

- IP router: 1 chassis (16.67) + 16 slot cards (9.17 x 16) + 32 10GELR port cards (4.20 x 32) + 32 OC192PoS (18.33 x 32) = 884.35; Normalized Price: 884.35/640 = 1.38 /Gbps
- TDM switch: 1 chassis (13.33) + 32 OC192 (1.67 x 32) + 32 10GE ((0.7+2x0.18) x (32/2)) = 83.73; Normalized Price: 83.73/640 = 0.13 /Gbps
- WDM switch: 1 chassis (2.5) + 12 degree, 40 channel (8.99 x 12) = 110.38; Normalized Price: 110.38/1600 = 0.069 / Gbps
- Fiber switch: 50/1920 = 0.026 / Gbps

It is more instructive to normalize each of these numbers with their switching bandwidth and then compare to the fiber switch (Table A.3). Clearly we see that there is a significant jump in going from optical to electrical switching fabrics. But even comparing the CRS-1 to a TDM switch which has a digital switching fabric, the former consumes 7 times the power and costs 10 times more.

As mentioned in Chapter 1, the objective of this exercise is not to say that the switches are equivalent because clearly they are not. They perform functions very differently. The objective is to say that there are some functions that circuits are exceedingly good at – like recovery, guarantees and on-demand-bandwidth; such that if we eliminate circuits and replace those functions with packets, we end up paying with higher operational and capital costs (as showed in Chapter 4).



	Fiber Switch	WDM Switch	TDM Switch	Packet Switch
	Glimmerglass IOS600	Fujitsu Flashwave 7500	Ciena CoreDirector	Cisco CRS-1
B/w	1	1	1	1
Power	1 W/Gbps	5	51	332
Volume	1 in^3/Gbps	4	41	65
Price	1 $/Gbps	3	5	53

Table A.3: Normalized Values

Appendix B

Switch API

Our switch-API is modeled on version 1.0 of the OpenFlow protocol for packet switches. It covers the components and basic functions of circuit switches based on switching time-slots, wavelengths and fibers. It also covers hybrid-switches with both packet and circuit interfaces and/or switching fabrics; and includes provisions for mapping packet-flows to circuit-flows. The entire API is presented in [36] – here we cover the main features.

Describing a Physical Circuit Port

```
struct ofp_phy_cport {
    uint16_t port_no;
    uint8_t hw_addr[OFP_ETH_ALEN];
    char name[OFP_MAX_PORT_NAME_LEN]; /* Null-terminated */

    uint32_t config;        /* Bitmap of OFPPC_* flags. */
    uint32_t state;         /* Bitmap of OFPPS_* flags. */

    /* Bitmaps of OFPPF_* that describe features.  All bits zeroed if
     * unsupported or unavailable. */
    uint32_t curr;          /* Current features. */
    uint32_t advertised;    /* Features being advertised by the port. */
    uint32_t supported;     /* Features supported by the port. */
    uint32_t peer;          /* Features advertised by peer. */

    /* Extensions for circuit switch ports */
    uint32_t supp_sw_tdm_gran; /* TDM switching granularity OFPTSG_* flags */
    uint16_t supp_swtype;      /* Bitmap of switching type OFPST_* flags */
    uint16_t peer_port_no;     /* Discovered peer's switchport number */
    uint64_t peer_datapath_id; /* Discovered peer's datapath id */
    uint16_t num_bandwidth;    /* Identifies number of bandwidth array elems */
    uint8_t  pad[6];           /* Align to 64 bits */
    uint64_t bandwidth[0];     /* Bitmap of OFPCBL_* or OFPCBT_* flags */

};
```

```
/* following capabilities are defined for circuit switches */
    OFPC_CTG_CONCAT     = 1 << 31, /* Contiguous concat on all TDM ports. */
    OFPC_VIR_CONCAT     = 1 << 30, /* Virtual concat on all TDM ports. */
    OFPC_LCAS           = 1 << 29, /* Link Capacity Adjustment Scheme */
    OFPC_POS            = 1 << 28, /* Packet over Sonet adaptation */
    OFPC_GFP            = 1 << 27, /* Generic Framing Procedure */
    OFPC_10G_WAN        = 1 << 26  /* native transport of 10G_WAN_PHY
                                      on OC-192 */
/* The following have been added for WAN interfaces */
    OFPPF_X          = 1 <<20, /* Dont care applicable to fiber ports */
    OFPPF_OC1        = 1 <<21, /* 51.84 Mbps OC-1/STM-0 */
    OFPPF_OC3        = 1 <<22, /* 155.52 Mbps OC-3/STM-1 */
    OFPPF_OC12       = 1 <<23, /* 622.08 Mbps OC-12/STM-4 */
    OFPPF_OC48       = 1 <<24, /* 2.48832 Gbps OC-48/STM-16 */
    OFPPF_OC192      = 1 <<25, /* 9.95328 Gbps OC-192/STM-64 */
    OFPPF_OC768      = 1 <<26, /* 39.81312 Gbps OC-768/STM-256 */
    OFPPF_100GB      = 1 <<27, /* 100 Gbps */
    OFPPF_10GB_WAN   = 1 <<28, /* 10 Gbps Ethernet WAN PHY (9.95328 Gbps) */
    OFPPF_OTU1       = 1 <<29, /* OTN OTU-1 2.666 Gbps */
    OFPPF_OTU2       = 1 <<30, /* OTN OTU-2 10.709 Gbps */
    OFPPF_OTU3       = 1 <<31  /* OTN OTU-3 42.836 Gbps */
```

Specifying a Cross-Connection

```
struct ofp_connect {
  uint16_t wildcards;                /* identifies ports to use below */
  uint16_t num_components;           /* identifies number of cross-connects
                                        to be made - num array elems */
  uint8_t  pad[4];                   /* Align to 64 bits */

  uint16_t in_port[0];               /* OFPP_* ports - real or virtual */
  uint16_t out_port[0];              /* OFPP_* ports - real or virtual */

  struct ofp_tdm_port in_tport[0];   /* Description of a TDM channel */
  struct ofp_tdm_port out_tport[0];

  struct ofp_wave_port in_wport[0];  /* Description of a Lambda channel */
  struct ofp_wave_port out_wport[0];
};
```

Specifying a TDM port

```
struct ofp_tdm_port {

  uint16_t tport;      /* port numbers in OFPP_* ports */
  uint16_t tstart;     /* starting time slot */
  uint32_t tsignal;    /* one of OFPTSG_* flags */
};
```

/* Minimum switching granularity of TDM physical ports available in a datapath. */

```
enum ofp_tdm_gran {
    OFPTSG_STS_1,        /* STS-1    / STM-0    */
    OFPTSG_STS_3,         /* STS-3    / STM-1    */
    OFPTSG_STS_3c,        /* STS-3c   / STM-1    */
    OFPTSG_STS_12,        /* STS-12   / STM-4    */
    OFPTSG_STS_12c,       /* STS-12c  / STM-4c   */
    OFPTSG_STS_48,        /* STS-48   / STM-16   */
    OFPTSG_STS_48c,       /* STS-48c  / STM-16c  */
    OFPTSG_STS_192,       /* STS-192  / STM-64   */
    OFPTSG_STS_192c,      /* STS-192c / STM-64c  */
    OFPTSG_STS_768,       /* STS-768  / STM-256  */
    OFPTSG_STS_768c       /* STS-768c / STM-256c */
};
```

Specifying a lambda channel

```
struct ofp_wave_port {
  uint16_t wport;      /* restricted to real port numbers in OFPP_* ports */
  uint8_t  pad[6];     /* align to 64 bits */
  uint64_t wavelength; /* use of the OFPCBL_* flags */
};
```

```
enum ofp_port_lam_bw {
    OFPCBL_X        = 1 << 0,/* 1 if fiber switch, 0 if wavelength switch */
    OFPCBL_100_50  = 1 << 1,/* 1 if 100GHz channel spacing, 0 if 50GHz */
    OFPCBL_C_L      = 1 << 2,/* 1 if using C-band frequencies, 0 if L-band */
    OFPCBL_OSC      = 1 << 3,/* 1 if supporting OSC at 1510nm, 0 if not */
    OFPCBL_TLS      = 1 << 4,/* 1 if using a TLS, 0 if not */
    OFPCBL_FLAG     = 1 << 5 /* 1 if reporting wave-support, 0 if reporting used
waves
};
```

Circuit flow setup, modification and teardown (controller-> datapath)

```
struct ofp_cflow_mod {

  struct ofp_header header;
  uint16_t command;             /* one of OFPFC_* commands */
  uint16_t hard_timeout;        /* max time to connection tear down,
                                   if 0 then explicit tear-down required */
  uint8_t pad[4];
  struct ofp_connect connect;    /* 8B followed by variable length arrays */
  struct ofp_action_header actions[0]; /* variable number of actions */
};
```

Action structure for OFPAT_CKT_OUTPUT, which sends packets out of a circuit port

```
struct ofp_action_ckt_output {
  uint16_t type;                   /* OFPAT_CKT_OUTPUT */
  uint16_t len;                    /* Length is 24 */
  uint16_t adaptation;             /* Adaptation type - one of OFPCAT_* */
  uint16_t cport;                  /* Real or virtual OFPP_* ports */

  /* Define the circuit port characteristics if necessary */
  uint64_t wavelength;             /* use of the OFPCBL_* flags */
  uint32_t tsignal;                /* one of the  OFPTSG_* flags. Not valid if
```

```
                                       used with ofp_connect for TDM signals */
  uint16_t tstart;                  /* starting time slot. Not valid if used
                                       with of_connect for TDM signals */
  uint16_t tlcas_enable;            /* enable/disable LCAS */
};
```

Port Status: physical circuit port has changed in the datapath

```
struct ofp_cport_status {
    struct ofp_header header;
    uint8_t reason;          /* One of OFPPR_*. */
    uint8_t pad[7];          /* Align to 64-bits. */
    struct ofp_phy_cport desc;
};
```

Appendix C

Network API

We present a sampling of the network-API we used in the prototypes discussed in Chapters 2, 3 and 5.

```
// Create a virtual port as an interface between packet and circuit
// switching domains
void createVirtualPort(uint64_t switchId, uint16_t vportId);

// Update virtual port with circuit-flows defined by ofp_tdm_port.
// @insert  true adds circuits to the vport, false deletes them.
// @cflowId caller-defined ID to identify circuits
void updateVcgWithCflow(uint64_t switchId, uint16_t vportId,
                       struct ofp_tdm_port& tpt, bool insert, uint32_t cflowId );

// Update virtual port with packet-flows defined by ofp_match
// Currently bidirectional packet flows are created such that:
//  -- vlan tags are added to packet that match before being forwarded
//     out of the virtual port
//  -- van tags are matched in the reverse direction along with virtual-ports as
//     in_port
// @insert  true adds permanent packet-flows to the vport, false deletes them.
// @cflowId caller-defined cookie to identify permanent flows
void updateVcgWithPflow(uint64_t switchId, uint16_t vportId, ofp_match pflow,
                        uint16_t vlanid, bool insert, uint64_t cookie );

// Poll port stats for virtual port
void pollPortStatsReq(uint64_t switchId, uint16_t vportId);

// Increase bandwidth along the path defined in Route.
// @ addBw bandwidth to add in Mbps. If bandwidth is quantized (eg. time-slots)
//   the closest value not greater than addBw will be used
void increaseBandwidth(uint32_t addBw, Route& path);

// Increase bandwidth along the path defined in Route.
void decreaseBandwidth(uint32_t delBw, Route& path);
```

```
// Create a bidirectional cross-connection defined in tpt1 and tpt2
// as part of the circuit flow identified  by cflowId
void addXconn(uint64_t switchId, struct ofp_tdm_port& tpt1,
              struct ofp_tdm_port& tpt, uint32_t cflowId );

// Delete the bidirectional cross-connection identified by cflowId
void deleteXconn(uint64_t switchId, uint32_t cflowId );

// Setup a 'flow' along a non-MPLS_tunnel 'route' - thus the definition
// of a flow is the same at each switch along the route. The only action
// applied is OFPAT_OUTPUT. Everything in Flow should be in network byte order.
// @nw_inport and @nw_outport refer to the ports where packet enters and exits
// the network, and should be specified in host byte order
bool setupFlowInIPRoute(const Flow& flow, const Routing_module::Route& route,
                        uint16_t nw_inport, uint16_t nw_outport,
                        uint16_t flow_timeout );

// Setup a 'flow' along a 'route' which includes an MPLS tunnel.
// Instead of the information passed in @'route', we use
//   -- @srcBr (tunnel-head) label information @srcOutLabel is pushed and outport
//      @srcOp is used
//   -- @dstBr (tunnel-tail) the flow definition changes to the physical in-port
//      @dstInPort
// Everything in Flow should be in network byte order.
// nw_inport and nw_outport refer to the ports where packet enters and exits the
// network, and should be specified in host byte order
bool setupFlowInTunnelRoute(const Flow& flow, const Routing_module::Route& route,
                            uint16_t nw_inport, uint16_t nw_outport,
                            uint16_t flow_timeout, datapathid srcBr,
                            uint16_t srcOp, uint32_t srcOutLabel,
                            datapathid dstBr, uint16_t dstInPort);

// Setup an LSP - we do not support LSP paths with size < 2 - ie LSP must
// traverse at least 3 nodes including headend and tailend switch. We always do
// Penultimate Hop Popping due to lack of multiple tables in switch. Associate
// tunnelId tide with TE tunnel
bool setupLsp(Cspf::RoutePtr& route, tunn_elem& te, uint16_t payload_ethtype,
              tunnelId tid);

// LSP flow entries are removed in switches between the head and tail ends.
// flow entries in the head and tail ends will idle-timeout when the
// lsp is no longer used - we assume this is true when this function is called
bool teardownLsp(tunnelId tid, uint16_t payload_ethtype);


// To re-route flow over tunnel, sends a flow_mod message
// to head-end and possibly tail-end routers
void rerouteFlows(tunnelId newtid, datapathid srcBr, datapathid dstBr,
                  tunnelId oldtid);

// Route flow only over IP links (no tunnel-links)
void routeFlow(Flow& flow, datapathid br);

// Asks for flow stats from switch for the corresponding tunnel
// Does not request from tunnel head-end as we do not know what is matching
// and going into the tunnel. Thus it asks the switch that is the next-hop
// from the tunnel head-end.
void sendTunnelStatsReq(tunnelId tid);

// Given an explicit route in 'route', verifies that a route can be found
// along the explicit path which meets the bandwidth and priority constraints.
// Note that the caller may only specify the datapathids in 'route' and not the
// ports. In that case, the function populates the outport and inport for each
// link in the 'route'. If this route were to be installed, the caller MUST
// then submit a call to get_route. If this route could potentially eject
// some lower priority tunnels, then the tunnel_ids of those tunnels are reported
// in 'eject'. This function does NOT assume that the 'route' is eventually
// installed, nor does it assume the 'eject's are actually ejected.
// Finally, like get_route, this function is applicable only to routes for
// new tunnels, NOT existing ones. After checking, if the caller wishes to
// enable this new tunnel route, they can use get_route
bool check_explicit_route(RoutePtr& route, uint32_t resBw, uint8_t priority,
                          std::vector<uint16_t>& eject);

// Given a tunnel-id for an existing tunnel together with it's existing route
// (including port numbers), current reserved bandwdith and priority, this
// function checks if a new_reserved bandwidth is possible along the same route.
// If it is possible, the function returns true.
// If it is possible, but requires ejecting other lower priority tunnels, the
// function returns true and populates the would-be-ejected tunnel_ids
// in 'eject'.
// If it is possible, but requires using a new route, the function returns true,
// clears the 'route', populates the new 'route', and sets 'newroute' to true.
// If additionally the new 'route', bumps off lower priority tunnels, their
// tunnel_ids are returned in 'eject'
// This is a check - the function does not assume that the bandwidth reservation
// changes or new routes are installed or any tunnels are actually ejected.
bool check_existing_route(RoutePtr& route, uint32_t currResBw, uint8_t priority,
                          tunnelId tid, uint32_t newResBw, bool& newroute,
                          std::vector<uint16_t>& eject);

// In response to check_existing_route, the caller may decide to alter the
// existing route. The caller informs Cspf of the changes using
// set_existing_route. 'route is the existing route (when check_existing_route
// was called). if the route has since changed, then newroute is true and the new
// route is in 'new_route'. If the caller
// ejected routes to establish either the 'newResBw', or the 'new_route', it
// informs Cspf of the ejected tunnels in 'ejected'
void set_existing_route(uint16_t tid, uint8_t priority, uint32_t currResBw,
                        uint32_t newResBw, RoutePtr& route, bool newroute,
                        RoutePtr& new_route, std::vector<uint16_t>& ejected);

// rerouteEjectedTunnels is called whenever a call to Cspf::get_route returns
// tunnel ids of LSPs ejected due to CSPF routing with priorities. Tunnels may
// also be ejected as a result of auto-bandwidth. This function performs the
// following sequence of steps:
// 1) It asks Cspf::get_route for a new route for the ejected tunnel
// 2) It creates a new LSP over the new route and gives it a new tunnel-id and
// updates the Rib and tunnel_db with characteristics of the old ejected tunnel
// 3) moves flows from the ejceted-LSP to the newly created one
// 4) removes (un-installs) the ejected LSP from the data-plane
void rerouteEjectedTunnels(tunnId etid);
```

Appendix D

Lines of Code Comparison

In this Appendix, we present the details of the Lines-of-Code analysis, the results of which were discussed in Chapter 3.

Table D.1 presents the output of CLOC for two open-source projects that have implemented OSPF-TE and RSVP-TE. The former is from the Quagga project [63]. Note that we have only accounted for the ospfd implementation, not any of the other routing protocols that are part of Quagga. Also the Quagga suite does not include RSVP-TE. We use the IST-Tequila project implementation for RSVP-TE [64, 65].

quagga-0.99.18/ospfd

```
-------------------------------------------------------------------------------
Language            files     blank        comment        code
-------------------------------------------------------------------------------
C                      26      6650           4276       32046
C/C++ Header           24       559            923        2198
make                    1        12              1          25
-------------------------------------------------------------------------------
SUM:                   51      7221           5200       34269
-------------------------------------------------------------------------------
```

rsvpd.0.70-rc2/rsvpd

```
-------------------------------------------------------------------------------
Language            files     blank        comment        code
-------------------------------------------------------------------------------
C                      78      7875           9866       45740
C/C++ Header           55      1297           2043        4243
```

make	7	125	71	302
Bourne	1	19	20	151
SUM:	141	9316	12000	50436

Table D.1: IP/MPLS Control Plane Implementation

The UNI is typically instantiated using a signaling protocol like RSVP-TE with suitable extensions [22]. Table D.2 shows an implementation of the OIF UNI from the IST-MUPBED project [66, 67].

rsvp-agent

Language	files	blank	comment	code
Bourne	8	3045	4334	23500
C++	21	1080	858	8006
m4	1	770	20	6206
Java	30	626	748	1499
C/C++ Header	20	257	501	1177
Python	2	153	380	683
make	3	5	6	13
Bourne	1	1	0	2
SUM:	86	5937	6847	41086

Table D.2: UNI Implementation

Finally the GMPLS protocols for the transport network control plane include further extensions of the MPLS control plane protocols – OSPF-TE and RSVP-TE. Such extensions have been standardized [20] and Table D.3 shows implementations of the GMPLS protocols by the DRAGON project [68, 69]. Note that in accounting for the latter, we are ignoring the code required for the proprietary interfaces in the transport network vendor islands.

dragon-sw/zebra/ospfd

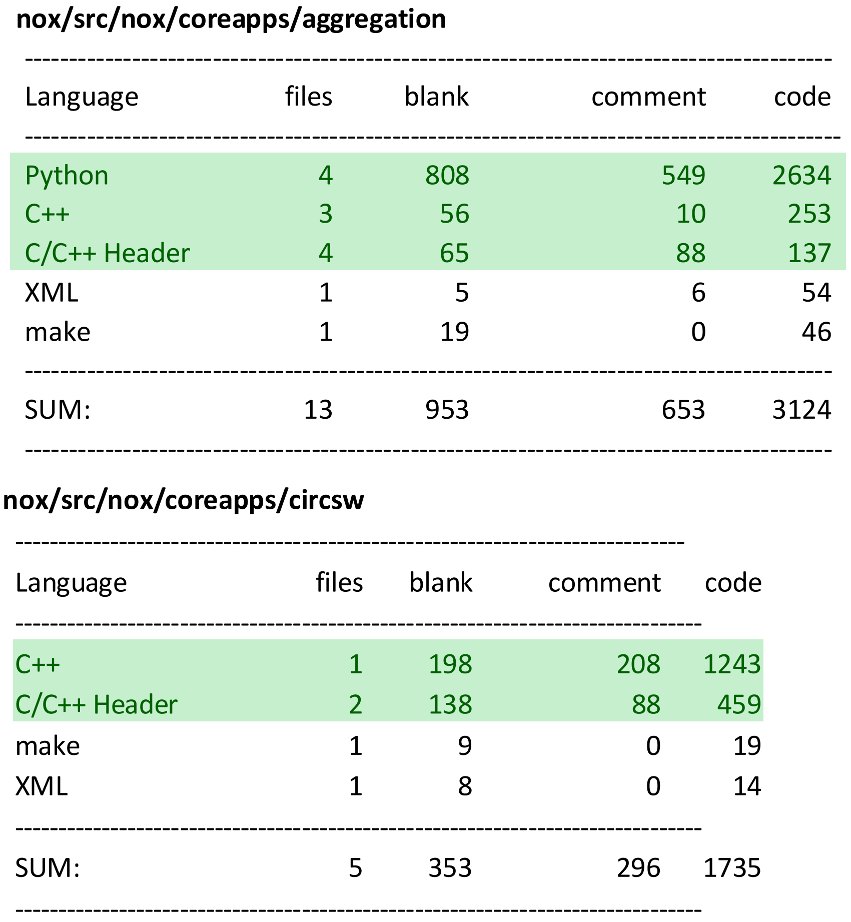
```
-------------------------------------------------------------------------------
Language          files     blank     comment          code
-------------------------------------------------------------------------------
C                 30        7113      4102             35514
C/C++ Header      26        618       1014             2522
make              1         9         1                36
-------------------------------------------------------------------------------
SUM:              57        7740      5117             38072
-------------------------------------------------------------------------------
```

dragon-sw/kom-rsvp

```
-------------------------------------------------------------------------------
Language          files     blank     comment          code
-------------------------------------------------------------------------------
C++               85        4384      3745             31004
C/C++ Header      106       2395      2733             11851
Bourne            10        1299      1313             11146
C                 2         128       128              821
Tcl/Tk            5         123       146              644
yacc              2         119       48               642
m4                1         20        57               476
make              12        92        3                344
Java              9         37        66               227
lex               2         37        46               109
ASP.Net           1         5         0                41
Perl              1         1         0                18
-------------------------------------------------------------------------------
SUM:              236       8640      8285             57323
-------------------------------------------------------------------------------
```

Table D.3: GMPLS Control Plane Implementation

In Table D.4, to implement the network function using our control solution requires only 4726 lines-of-code – *two orders of magnitude lesser* than the industry solution.



nox/src/nox/coreapps/aggregation

```
-------------------------------------------------------------------------------
Language                files          blank          comment         code
-------------------------------------------------------------------------------
Python                      4            808              549         2634
C++                         3             56               10          253
C/C++ Header                4             65               88          137
XML                         1              5                6           54
make                        1             19                0           46
-------------------------------------------------------------------------------
SUM:                       13            953              653         3124
-------------------------------------------------------------------------------
```

nox/src/nox/coreapps/circsw

```
--------------------------------------------------------------------
Language                  files     blank        comment     code
--------------------------------------------------------------------
C++                           1       198            208     1243
C/C++ Header                  2       138             88      459
make                          1         9              0       19
XML                           1         8              0       14
--------------------------------------------------------------------
SUM:                          5       353            296     1735
--------------------------------------------------------------------
```

Table D.4: Control-Function Implementation with Unified Control Architecture

The base code of NOX is roughly 67,700 lines of code (Table D.5 and [51]) to which we have added 1100 lines to support circuit-switching - OpenFlow extensions for circuits, a discovery module and a circuit-API. The Quagga software suite is based on the zebra-daemon and related libraries. A lines-of-code estimate for this base code is shown in Table D.6, where we have *not* included the protocol implementations for all the protocols found in the software suite (OSPF, OSPFv6, ISIS, RIP, RIPnG, and BGP as well as vtysh, ospfclient, tools directories are not included).

Language	files	blank	comment	code
C++	161	6129	5647	37161
C/C++ Header	217	4806	11106	16190
Python	125	3623	5140	13516
make	60	522	63	2004
m4	26	117	14	1542
Bourne	40	180	231	1262
C	3	138	142	901
CSS	1	25	10	462
Bourne Again Shell	3	34	77	360
SQL	1	37	104	203
Perl	2	18	0	178
HTML	4	24	1	145
XSD	2	6	0	50
SUM:	645	15659	22535	73974

Table D.5: NOX Base Lines of Code

Language	files	blank	comment	code
C	95	9137	7098	48255
Bourne	10	4151	6911	25102
m4	7	1044	313	9275
C/C++ Header	52	1159	2164	3573
Bourne	8	67	85	351
make	10	75	55	241
awk	2	37	89	129
SUM:	184	15670	16715	86926

Table D.6: Quagga Base Lines of Code

Bibliography

[1] Autonomous System: http://en.wikipedia.org/wiki/Autonomous_system_(Internet)

[2] Jennifer Rexford. Evolving Towards a Self Managing Network. Self Managing Networks Summit. June 2005. http://research.microsoft.com/en-us/um/redmond/events/smnsummit/techprogram.aspx

[3] ITU-T, Generic Functional Architecture of Transport Networks, Recommendation G.805, 2000.

[4] Greg Bernstein, Bala Rajagopalan, Debanjan Saha. Optical Network Control. ISBN 0201753014, 2004.

[5] Vishnu Shukla. Optical Control Plane Deployment – Lessons Learned.OIF Workshop on ASON/GMPLS Implementations, Oct 2006 http://www.oiforum.com/public/documents/061016-Verizon.pdf

[6] Tier 1 and Tier 2 ISPs: http://en.wikipedia.org/wiki/Tier_1_network

[7] Cisco Visual Networking Index: Forecast and Methodology, 2010 – 2015. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[8] Personal communication with Curtis Villamizar, Infinera.

[9] Charles Fraleigh, Fouad Tobagi, and Christophe Diot. Provisioning IP Backbone Networks to Support Latency Sensitive Traffic. Infocomm 2003.

[10] Benjamin Chen, Fouad Tobagi. Optical network design to minimize switching and transceiver equipment costs. Optical Switching and Networking 2009.

[11] Saurav Das, Guru Parulkar, Nick McKeown. Unifying Packet and Circuit Switched Networks. Below IP Networking Workshop in association with Globecom, November 2009.

[12] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM Computer Communication Review, Volume 38, Number 2, April 2008.

[13] Saurav Das, Guru Parulkar, Nick McKeown. Simple Unified Control for Packet and Circuit Networks. Future Global Networks Workshop, IEEE Photonics Society Summer Topicals, June 2009.

[14] OpenFlow http://www.openflow.org/index.php

[15] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casado, Nick McKeown, Scott Shenker. NOX: Towards an Operating System for Networks. ACM SIGCOMM Computer Communication Review, July 2008.

[16] Nick McKeown. Software Defined Network and OpenFlow. Structure 2010. http://gigaom.com/2010/06/23/structure-2010-reinventing-the-internet-get-ready-for-software-defined-networks/

[17] Scott Shenker, Martin Casado, Teemu Koponen, Nick McKeown et al. The Future of Networking and the Past of Protocols. June 2011.
http://www.slideshare.net/martin_casado/sdn-abstractions?from=ss_embed

http://opennetsummit.org/talks/shenker-tue.pdf

[18] Martin Casado. Architectural Support for Security Management in Enterprise Networks. PhD Thesis, Stanford University, August 2007.

[19] E. Mannie. Generalized Multiprotocol Label Switching (GMPLS) Architecture. IETF RFC 3945, Oct. 2004.

[20] IETF CCAMP working group http://datatracker.ietf.org/wg/ccamp/charter/

[21] ITU-T. Architecture for the Automatically Switched Optical Network (ASON). Rec. G.8080/Y.1304, Nov. 2001 (and rev., Jun. 2006).

[22] Optical Internetworking Forum
http://www.oiforum.com/public/impagreements.html#UNI

[23] Adrian Farrel, Igor Bryskin. GMPLS Architecture and Applications. ISBN 0120884224, 2006.

[24] Ayan Banerjee, John Drake, Jonathan Lang, Brad Turner, Daniel Awduche, Lou Berger, Kireeti Kompella, Yakov Rekhter. Generalized Multiprotocol Label Switching: An Overview of Signaling Enhancements and Recovery Techniques. IEEE Communications Magazine, July 2001.

[25] Ayan Banerjee, John Drake, Jonathan Lang, Brad Turner, Kireeti Kompella, Yakov Rekhter. Generalized Multiprotocol Label Switching: An Overview of Routing and Management Enhancements. IEEE Communications Magazine, January 2001.

[26] Monique J. Morrow, Mallik Tatipamula, Adrian Farrel. GMPLS: The Promise of the Next-Generation Optical Control Plane. Guest Editorial, IEEE Communications Magazine, July 2005.

[27] Opinion: http://www.lightreading.com/document.asp?doc_id=208072

[28] OpenFlow Switch Specification v1.0
http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf

[29] Saurav Das, Guru Parulkar, Nick McKeown. Why OpenFlow/SDN Can Succeed Where GMPLS Failed. European Conference on Optical Communications (ECOC), September 2012.

[30] Spectral grids for WDM applications: ITU-T G.694.1
http://www.itu.int/rec/T-REC-G.694.1-200206-I/en

[31] Link Layer Discovery Protocol (LLDP) IEEE 802.1AB

[32] D. Katz, D. Ward. Bidirectional Forwarding Detection (BFD). RFC 5580.

[33] Infinera DTN: http://www.infinera.com/products/dtn.html

[34] Fujitsu Flashwave 9500: http://www.fujitsu.com/us/services/telecom/products/

[35] pac.c webpage: http://www.openflow.org/wk/index.php/PAC.C

[36] Experimental extensions to the OpenFlow v1.0 in support of circuit switching (v0.3)
http://www.openflow.org/wk/images/8/81/OpenFlow_Circuit_Switch_Specification_v0.3.pdf

[37] Vinesh Gudla, Saurav Das, Anujit Shastri, Guru Parulkar, Shinji Yamashita, Leonid Kazovsky, Nick McKeown. Experimental Demonstration of OpenFlow Control of

Packet and Circuit Switches. Optical Fiber Communications / National Fiber Optics Engineers Conference (OFC/NFOEC), March 2010.

[38] Demo video: OpenFlow control of packet and wavelength switches http://openflow.smugmug.com/OpenFlow-Videos/pacc-demos/11583800_tCEwT#895139512_E8jA3-A-LB

[39] OpenFlow Reference Switch: http://www.openflow.org/wp/downloads/

[40] Quad-Port Gigabit NIC http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Guide#Reference_NIC_Walkthrough_

[41] NetFPGA website: http://netfpga.org/

[42] OpenFlow v1.0 Implementation in NetFPGA http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/OpenFlowNetFPGA100

[43] Trendnet GE-SFP convertor http://www.trendnet.com/products/proddetail.asp?prod=110_TFC-1000MGB&cat=22

[44] Ciena CoreDirector: http://www.ciena.com/products/core-director/

[45] SuperComputing2009 demo: http://www.openflow.org/wp/2009/11/openflow-demo-at-sc09/

[46] Saurav Das, Guru Parulkar, Preeti Singh, Daniel Getachew, Lyndon Ong, Nick McKeown. Packet and Circuit Network Convergence with OpenFlow. Optical Fiber Communications / National Fiber Optics Engineers Conference (OFC/NFOEC), March 2010.

[47] ENVI http://www.openflow.org/wp/gui/

[48] Quanta LB4G/ Pronto 3240: http://www.prontosys.net/test/pronto3240.htm

[49] Indigo: OpenFlow for Hardware Switches http://www.openflowhub.org/display/Indigo/Indigo+-+OpenFlow+for+Hardware+Switches

[50] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, Guru Parulkar. Can the Production Network Be the Testbed? Operating System Design and Implementation (OSDI), October 2010.

[51] NOX distribution: http://noxrepo.org/noxwiki/index.php/Main_Page

[52] GEC 8 http://groups.geni.net/geni/wiki/GEC8DemoSummary

[53] Saurav Das, Yiannis Yiakoumis, Guru Parulkar, Preeti Singh, Dan Getachew, Premal Dinesh Desai, Nick McKeown, Application-Aware Aggregation and Traffic Engineering in a Converged Packet-Circuit Network, OFC/NFOEC, March 2011.

[54] Yiannis Yiakoumis, Dynamic Flow Aggregation http://www.youtube.com/watch?feature=player_embedded&v=nFzpdxl521Y

[55] http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

[56] Personal communication with Stuart Elby, Verizon.

[57] Christopher Metz. Ingredients for Better Routing? Read the Label. IEEE Internet Computing, Vol. 2, Issue 5, September/October 1998.

[58] Daniel Awduche, Yakov Rekhter. Multiprotocol Lambda Switching: Combining MPLS Traffic Engineering Control with Optical Crossconnects. IEEE Communications Magazine, March 2001.

[59] Lines of code wiki http://en.wikipedia.org/wiki/Source_lines_of_code

[60] CLOC http://cloc.sourceforge.net/

[61] PBR http://en.wikipedia.org/wiki/Policy-based_routing

[62] RFC4124 – DiffServ-aware MPLS Traffic Engineering

[63] Quagga project: http://www.quagga.net/about.php

[64] IST Tequila project: http://www.ist-tequila.org/

[65] RSVP-TE Daemon: http://dsmpls.atlantis.ugent.be/

[66] IST-MUPBED project: http://www.ist-mupbed.org/

[67] GMPLS UNI: http://sourceforge.net/projects/rsvp-agent/

[68] DRAGON: http://dragon.maxgigapop.net/twiki/bin/view/DRAGON/WebHome

[69] DRAGON LSR: http://dragon.east.isi.edu/twiki/bin/view/DRAGON/VLSR

[70] http://www.networkworld.com/news/2008/041708-cisco-juniper-operating-systems.html

[71] http://www.openflow.org/wk/index.php/Aggregation_on_a_Converged_Packet-Circuit_Network

[72] http://www.oiforum.com/public/documents/OIF-UNI-02.0-Common.pdf

[73] Saurav Das. Virtualizing the Transport Network - Why it Matters and How OpenFlow Can Help. OFELIA Workshop on OpenFlow extensions towards multi-layer and multi-domain networks. European Conference on Optical Communications (ECOC), September 2011.

[74] http://www.networkworld.com/news/2007/051607-cisco-routers-major-outage-japan.html

[75] Jean-Philipe Vasseur, Mario Pickavet, Piet Demeester. Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS. ISBN 012715051X, 2004.

[76] Personal communication with Ori Gerstel, Cisco.

[77] Guangzhi Li, Dongmei Wang, Jennifer Yates, Robert Doverspike, Charles Kalmanek. IP over Optical Cross-Connect Architectures. IEEE Communications Magazine, February 2007.

[78] Rocketfuel Project: http://www.cs.washington.edu/research/networking/rocketfuel/

[79] Jane Simmons. Optical Network Design and Planning. ISBN-10: 0387764755, 2008.

[80] KMI Research. North American National and Regional Fiber Optic Long Haul Routes Planned and In Place. June 2002.

[81] A. Medina, N. Taft, K. Salamatian, S. Bhattacharya, C. Diot. Traffic Matrix Estimation: Existing Techniques and New Directions. SIGCOMM 2002.

[82] Ralf Huelsermann, Matthias Gunkel, Clara Muesburger, Dominic Schupke. Cost Modeling and Evaluation of Capital Expenditures in Optical Multilayer Networks. Journal of Optical Networking, Vol. 7, No. 9, September 2008.

[83] R. Batchellor, O. Gerstel. Cost Effective Architectures for Core Transport Networks. Optical Fiber Communications Conference, OFC 2006.

[84] Sudipta Sengupta, Vijay Kumar, Debanjan Saha. Switched Optical Backbone for Cost-Effective Scalable Core IP Networks. IEEE Communications Magazine, June 2003.

[85] Ciena: http://www.lightreading.com/document.asp?doc_id=182351

[86] Juniper-NSN: http://www.lightreading.com/document.asp?doc_id=178407

http://www.lightreading.com/document.asp?doc_id=193988

[87] Alcatel-Lucent: http://www.lightreading.com/document.asp?doc_id=181849

[88] Ericsson: http://www.lightreading.com/document.asp?doc_id=187674

[89] http://www.heavyreading.com/details.asp?sku_id=2464&skuitem_itemid=1216

[90] http://www.lightreading.com/document.asp?doc_id=174032

[91] ITU Recommendation G.709. Interfaces for the Optical Transport Network

[92] Eric Osborne, Ajay Simha. Traffic Engineering with MPLS. ISBN-10: 1587050315, July 2002.

[93] ATM opinion: http://staff.washington.edu/gray/papers/whynotatm.html

[94] Guangzhi Li, Dongmei Wang, Charles Kalmanek, Robert Doverspike. Efficient Distributed Path Selection for Shared Restoration Connections. IEEE/ACM Transactions on Networking. Vol. 11. No. 5, Oct 2003.

[95] Matthieu Clouqueur, Wayne Grover. Quantitative Comparison of End-To-End Availability of Service Paths in Ring and Mesh Restorable Networks. National Fiber Optics Engineers Conference, NFOEC 2003 (see footnote #1)

[96] ODU0 and ODUflex: http://documents.exfo.com/appnotes/anote242-ang.pdf

[97] Pierre Francois, Clarence Filsfils, John Evans, Olivier Bonaventure. Achieving Sub-Second IGP Convergence in Large IP Networks. SIGCOMM CCR, Vol. 35, No. 2, July 2005.

[98] Panita Pongpaibool, Robert Doverspike, Matthew Roughan, Joel Gottlieb. Handling IP Traffic Surges via Optical Layer Reconfiguration. Optical Fiber Communications Conference, OFC 2002.

[99] Cisco CRS-1: http://www.cisco.com/en/US/prod/collateral/routers/ps5763/prod_brochure0900aecd800f8118.pdf

[100] Ciena 5400 series: http://www.ciena.com/products/category/packet-optical-switching/

[101] http://michaelbluejay.com/electricity/cost.html

[102] http://www.pge.com/tariffs/electric.shtml

[103] http://en.wikipedia.org/wiki/OAMP

[104] Operator Hourly Rate: http://www.payscale.com/research/US/Job=Network_Control_Technician/Hourly_Rate

[105] Rack Rentals: http://www.intercom.com/index.php?option=com_content&view=article&id=75&Itemid=85

[106] RFC 5036: LDP Specification

[107] MPLS with OpenFlow and Software Defined Networking http://www.openflow.org/wk/index.php/MPLS_with_OpenFlow/SDN

[108] Saurav Das, Ali Reza Sharafat, Guru Parulkar, Nick McKeown, MPLS with a Simple OPEN Control Plane, invited talk at Packet Switching Symposium at OFC/NFOEC'11, Los Angeles, March 2011.

[109] Ali Reza Sharafat, Saurav Das, Guru Parulkar, Nick McKeown, MPLS-TE and MPLS VPNs with OpenFlow, demonstration at SIGCOMM, Toronto, August 2011.

[110] Juniper Application Note: Using MPLS-Autobandwidth http://s-tools1.juniper.net/solutions/literature/app_note/350080.pdf

[111] RFC 4090: Fasr Re-Route extensions to RSVP-TE for LSP Tunnels

[112] RFC 2205: Resource Reservation Protocol (RSVP)

[113] RFC 5921: A Framework for MPLS in Transport Networks (MPLS-TP)

[114] RFC 5586: MPLS Generic Associated Channel

[115] RFC 2475: An Architecture for Differentiated Services.

[116] http://www.cisco.com/en/US/docs/ios/12_2s/feature/guide/fsdserv3.html

[117] OpenFlow Specification v.1.1 http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

[118] Open vSwitch: http://openvswitch.org/

[119] http://openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=PORTING;hb=HEAD

[120] http://yuba.stanford.edu/git/gitweb.cgi?p=movs.git;a=summary

[121] http://www.openflow.org/wk/index.php/OFTestTutorial

[122] http://yuba.stanford.edu/git/gitweb.cgi?p=moftest.git;a=summary

[123] http://yuba.stanford.edu/git/gitweb.cgi?p=mnox.git;a=summary

[124] Mininet: http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet

[125] Bob Lantz, Brandon Heller, and Nick McKeown. A Network on a Laptop: Rapid Prototyping for Software-Defined Networks. 9th ACM Workshop on Hot Topics in Networks, October 20-21, 2010, Monterey, CA

[126] http://en.wikipedia.org/wiki/MPLS-TP

[127] RFC 5654: Requirements of an MPLS-TP – Requirement #48

[128] Glenn Wellbrock. The Convergence of L1/2/3 Functionality in Next Generation Network Elements: A Carrier's Perspective. OFC/NFOEC 2011.

[129]Teemu Koponen, Martin Casado, Natasha gude, Jeremy Stirbling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, Scott Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. Operating Systems Design and Implementation (OSDI), October 2010.

[130] Martin Casado. Performance, Scalability and Reliability in OpenFlow Networks. A Primer…Stanford CS 244 lecture, Winter 2011.

[131] Amin Tootoonchian, Yashar Ganjali. A Distributed Control Plane for OpenFlow Networks. In Proceedings of NSDI Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN), April 2010.

[132] Jeffrey C. Mogul , Jean Tourrilhes , Praveen Yalagandula , Puneet Sharma , Andrew R. Curtis , Sujata Banerjee, DevoFlow: cost-effective flow management for high performance enterprise networks, Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks, p.1-6, October 20-21, 2010, Monterey, California.

[133]Nate Foster, Rob Harrison, Matthew L. Meola, Michael J. Freedman, Jennifer Rexford, and David Walker. Frenetic: A high-level language for OpenFlow Networks. Proc. Workshop on Programmable Routers for the Extensible Services of Tomorrow, December 2010.

[134] Open Networking Foundation (ONF) http://opennetworkingfoundation.org

[135] G. Wang, D. G. Andersen, M. Kaminsky, D. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan. c-Through: Part-time optics in data centers. In ACM SIGCOMM, Aug. 2010.

[136] N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. He- lios: A hybrid electrical/optical switch architecture for mod- ular data centers. In ACM SIGCOMM, Aug. 2010.

[137] Hamid Hajabdolali Bazzaz, Malveeka Tewari, Guohui Wang, George Porter, T. S. Eugene Ng, David G. Andersen, Michael Kaminsky, Michael A. Kozuch, Amin Vahdat. Switching the Optical Divide: Fundamental Challenges for Hybrid Electrical/Optical Datacenter Networks. ACM SOCC 2011.

[138] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Meyers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. "A clean slate 4D approach to network control and management," ACM SIGCOMM Computer Communications Review, October 2005

[139] Nick Feamster, Hari Balakrishnan, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe, "The case for separating routing from routers," Proc. ACM SIGCOMM workshop on Future Directions in Network Architecture, August 2004.

[140] Generic Switch Management Protocol (GSMP) http://datatracker.ietf.org/wg/gsmp/charter/

[141] RFC 3292: GSMP v3

[142] Forwarding and Control Element Separation (ForCES) http://datatracker.ietf.org/wg/gsmp/charter/

[143] Controller performance comparison http://www.openflow.org/wk/index.php/Controller_Performance_Comparisons