Emerging Applications: Networking Applications

Janani Ravi, Metha Jeeradit, John Kim

1. Introduction

Network is essentially a distributed information system providing access to shared data objects between networking applications running on different nodes. As the Internet gains its rapid growth, new and more demanding networking applications start to emerge such as voice over IP, storage over IP and media streaming. To cope with the demands of these applications, the underlying network processor needs to be designed efficiently.

2. Functionality and Requirements

As stated in the introduction, the main function of these networking applications is to communicate data, voice or video over a network. The requirements for these applications will depend on the interactive nature of the applications. For example, voice over IP applications require low delay, jitter and loss rate for the communication to be acceptable. Media streaming can tolerate moderate delay latency by buffering but still desire low jitter and loss rate. On the other hand, general TCP/IP routing does not pose any stringent requirements on delay, jitter or loss because the protocol is designed to tolerate them. In order to meet these requirements, the network processors must be flexible yet provide high performance.

In addition, the applications can be categorized in 2 categories [2]: control-plane tasks that are less time-critical and are responsible for maintaining device states and data-plane tasks which are core device operations that are responsible for processing, transmitting and receiving the packets.

3. General Characteristics

[2] states that networking applications usually include following properties:

- High Data-level parallelism
  - Amenable to multithreading technique
  - Multiple execution cores is desirable
- Well-define inputs: network packets
  - Need special purpose hardware to move packet data
- Modular in nature
  - Has a defined set of tasks to be performed
  - Can utilize a standardized organizations

In this section, we examine the characteristics of networking applications.

3.1 Computational-to-memory operations ratio

Figure 1 shows the instruction distributions for MiBench benchmark suite [3] containing a networking benchmark as its subset. Similarly, figure 2 shows the instruction distributions for a set of applications used in [4] that characterize different network processor architectures. From these two figures, there is a fair amount of computational operations to memory operation whereas branch operations occur very infrequently.

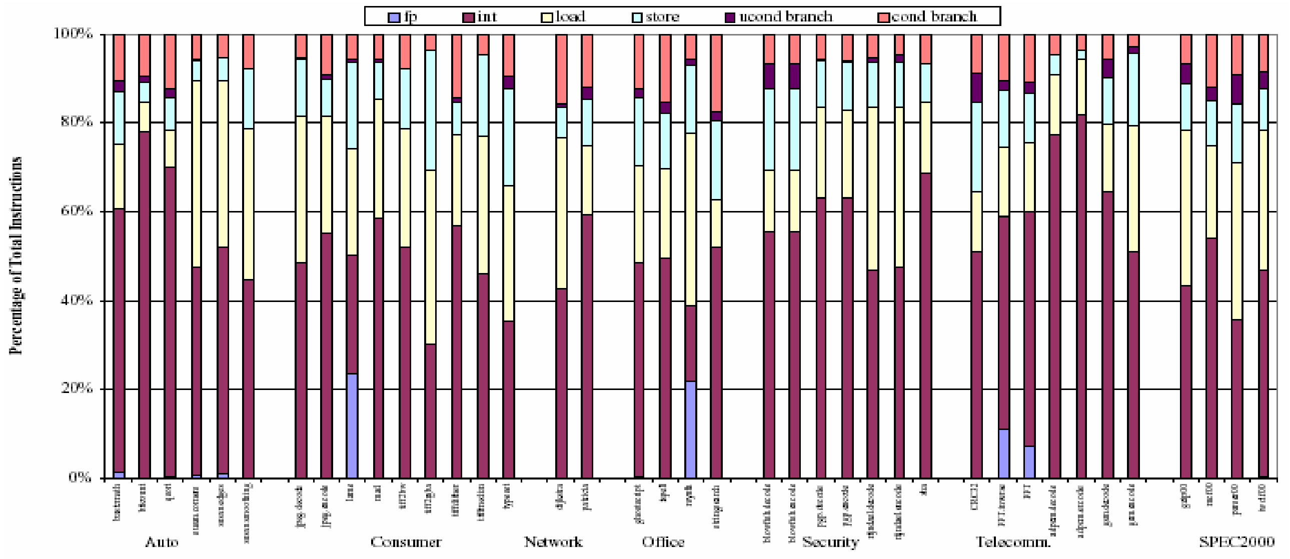
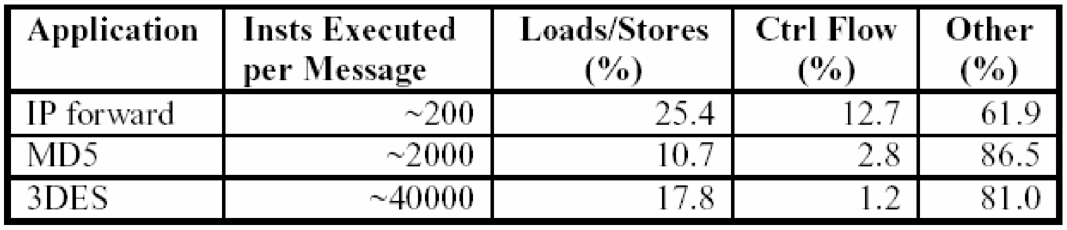


Figure 1: Dynamic Instruction Distribution for large data set.

Figure 1



Application	Insts Executed per Message	Loads/Stores (%)	Ctrl Flow (%)	Other (%)
IP forward	~200	25.4	12.7	61.9
MD5	~2000	10.7	2.8	86.5
3DES	~40000	17.8	1.2	81.0

Figure 2

1. IP forward performs address-based packet lookups in a tree data structure and is a component of conventional layer-3 switches and routers.
2. MD5 is a message digest application used to compute a unique signature over the data in each packet for authentication purposes.
3. 3DES is an encryption routine used in this domain to encrypt the full payload of a packet.

3.2 Working set size and memory access behavior

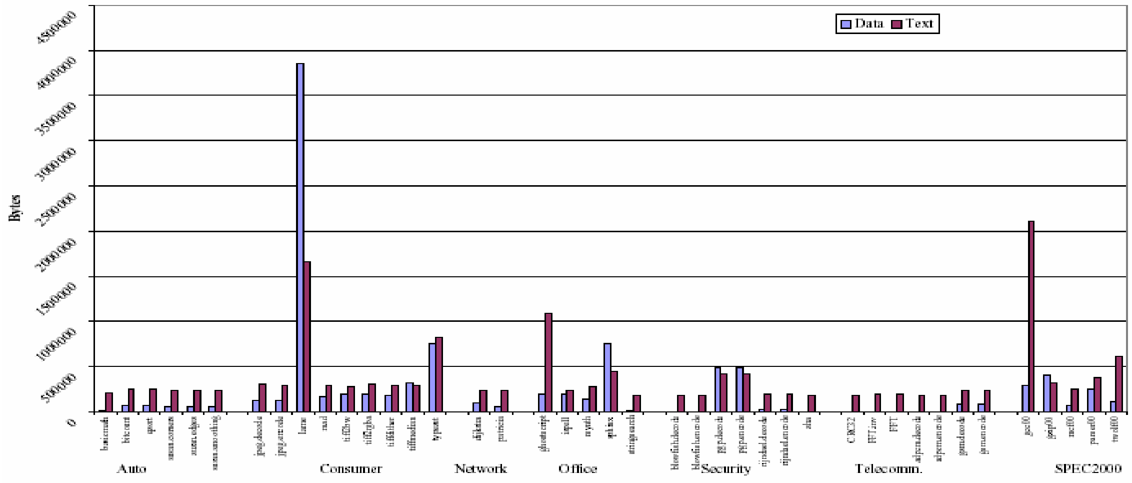
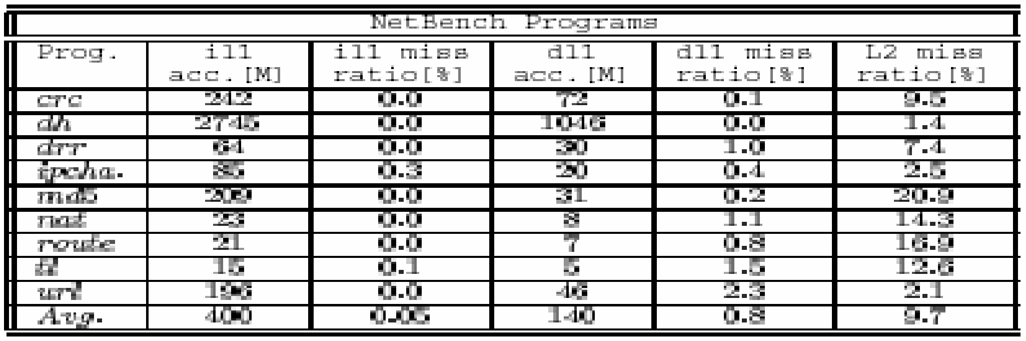


Figure 4: Text and data segment sizes. prediction rates for several schemes. **Figure 3**



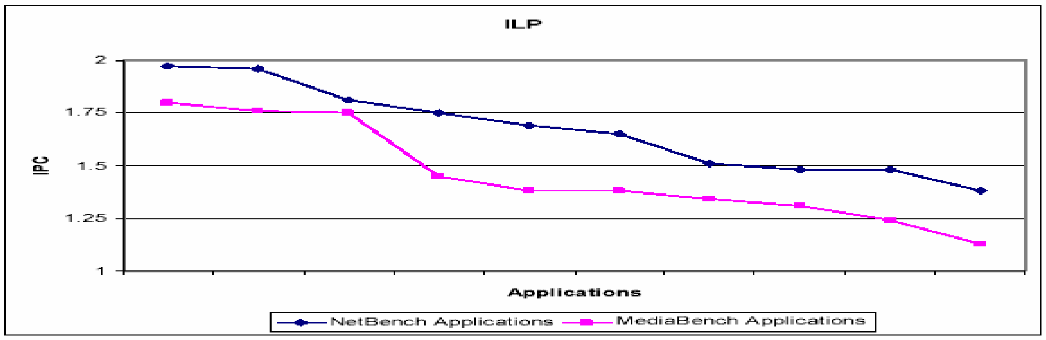
NetBench Programs					
Prog.	il1 acc. [M]	il1 miss ratio[%]	dl1 acc. [M]	dl1 miss ratio[%]	L2 miss ratio[%]
crc	242	0.0	72	0.1	9.5
dh	2745	0.0	1046	0.0	1.4
drr	64	0.0	30	1.0	7.4
ipcha.	85	0.3	20	0.4	2.5
md5	209	0.0	31	0.2	20.9
nat	23	0.0	8	1.1	14.3
route	21	0.0	7	0.8	16.9
tl	15	0.1	5	1.5	12.6
url	196	0.0	46	2.3	2.1
Avg.	400	0.05	140	0.8	9.7

Figure 4

Figure 3 shows the text and data sizes for the same MiBench suite showing the small data and text sizes of the networking benchmark. Figure 4 shows the caching behavior of another benchmark suite, Netbench [5]. It can be seen that both instructions and data are highly predictable as the global miss rate for both L1 and L2 caches are rather low.

3.3 ILP

ILP



- Average IPC: 1.67 for NetBench, 1.45 for MediaBench

Figure 5

As can be seen in Figure 5 (taken from [1]) there is not that much ILP available in networking applications.

4. Caches and Memory Access Patterns in Web Caching

In this section, we describe the caching behavior and memory access patterns for web caching in details. The idea of using proxy servers to cache came from the use of firewalls to filter the requests and responses of the users in organizations. Chances of the users in one organizations accessing the same set of documents is high and so cached documents at the server would be likely to result in hits.

Advantages:

- reduce bandwidth consumption
- reduce access latency
- reduce the workload of remote web servers
- improves availability of cached documents

Disadvantages:

- access to stale data
- extra processing due to cache miss
- single proxy cache would be a bottleneck

Desirable properties of WWW caching system: Fast access, robustness, transparency, scalability, efficiency, adaptivity, stability, load balancing, ability to deal with heterogeneity, simplicity.

4.1 Caching Architectures

Hierarchical
Caches at different levels such as bottom, institutional, regional and national

Advantages

- Shorter connections times-as copies are placed on intermediate nodes

Disadvantages

- Every level can introduce delays
- Higher level caches may introduce bottlenecks and have long queuing delays
- Multiple copies stored at different cache levels

Distributed
Caches only at the bottom level with sharing of data between these caches
Advantages

- Shorter transmission times-most of the traffic flows through less congested levels
- Better distributed bandwidth usage in the lower levels of network hierarchy

Disadvantages

- Higher bandwidth usage overall

Hybrid

There is a caching hierarchy with a certain number of caches cooperating at every level as in a distributed caching scheme.

In the following graphs
λ_{tot} - The relative popularity of the document larger indicates more popular
$\rho = 0.8$ This is the measure of network congestion
$k_c = 4$, The number of cooperating processors for hybrid caches

Connection Time

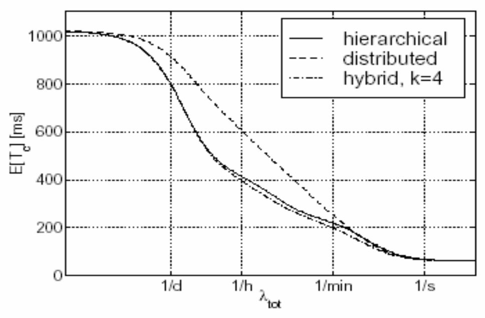


Figure 8: Connection time in a hybrid caching scheme with the optimal number of cooperating caches k_c.

Figure 6

This depends on the number of network links to reach the document that we need.
Unpopular documents: Both schemes experience very long connection times
Medium popularity: Hierarchical caches give better connection times
Very popular documents: Not much difference between the 2 schemes
This shows how a hybrid cache with 4 cooperating processors performs a little better than both the other schemes.

Transmission time

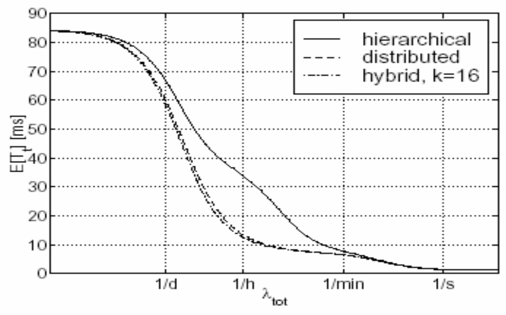


Figure 10: Transmission time for a hybrid caching scheme with the optimal number of cooperating caches k_t. National network is congested, $\rho = 0.8$.

Figure 7

This is the transmission times in a highly congested network scenario. Distributed routing has lower transmission times as documents travel over lower network levels. The hybrid scheme performs better than both the other schemes.

Total Latency

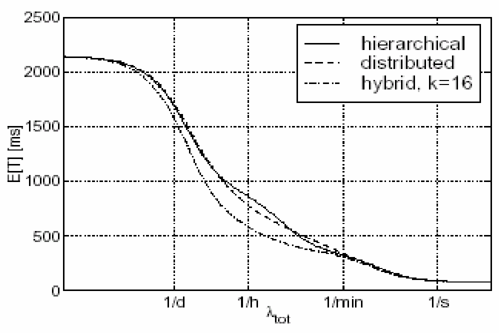


Figure 12: Total latency to retrieve a large document in a hybrid caching scheme with the optimum number of cooperating caches $k_{opt} = k_E = 16$. National network is congested $\rho = 0.8$. $S = 200$ KB.

Figure 8

Depending on the document size the connection time or the transmission time is more relevant to the total document latency. For small document sizes the connection time is more important than the transmission time and for larger documents the transmission time becomes more important. The optimum number of cooperating caches needed varies according to the document sizes. In this case it is 16.

4.2 Cache resolution/routing

The main challenge in these approaches is how to locate a cache containing a desired document quickly. This problem is difficult to scale as, unlike conventional routing protocols, it can't take advantage of route aggregation due to hierarchical addressing. Also the cache documents have to be updated frequently. This can be done using a cache routing table or hashing functions.(??? do we need to specify more details here)

4.3 Prefetching

Prefetching can be divided into 2 categories. Local and server-hint prefetching based on where the information for determining which objects to prefetch is generated.

In local prefetching the agent doing the prefetching(e.g. browser client or proxy) uses local information(e.g. reference patterns) to determine which objects to prefetch.

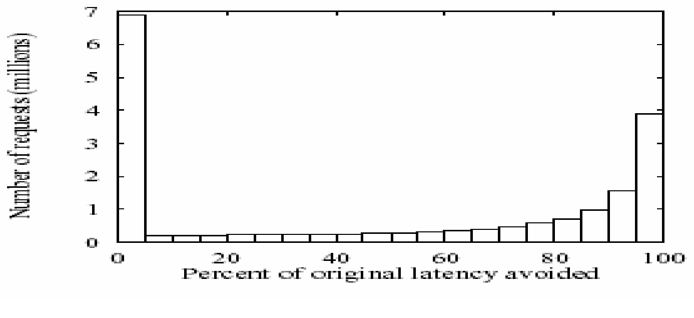
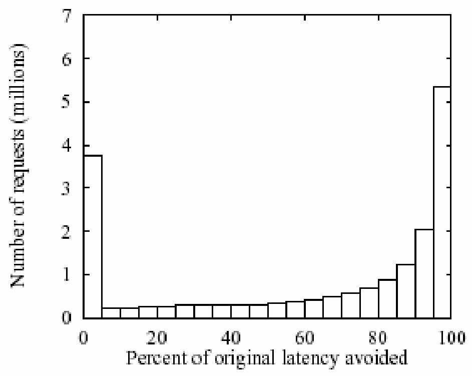
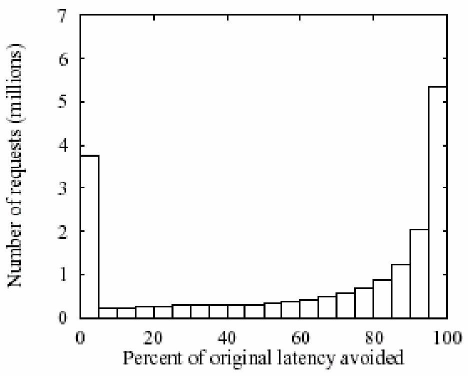


Figure 9

In server hint-based prefetching the server is able to use its content specific knowledge of the objects requested as well as the reference patterns from a far greater number of clients to determine which objects should be prefetched.



(c) Server-Hint Based without Caching



(d) Server-Hint Based with Caching

Figure 10

These graphs show a bimodal distribution where we can see significant latency reductions(peaks around 95% to 100%) or little to no reduction(peak at around 0%). Thus for these models a web request can see a significant latency reduction or little or no reduction.

4.4 Cache placement/ replacement

These policies try to minimize various cost metrics such as hit rate, byte hit rate, average latency and total cost. They can be categorized as follows:

1. Traditional replacement policies and its direct extensions-Least Recently Used, Least Frequently used, Pitkow/Rector which evicts objects in LRU order but if all have been accessed on the same day then the largest one is removed.
2. Key based replacement policies-Evicts objects based on size, LRU-MIN which favors smaller objects, LRU-Thresholds which behaves like LRU except that objects larger than a particular size are never cached, Lowest Latency First evict document with the lowest download latency and others.
3. Cost Based Replacement Policies-GreedyDual-Size associates a cost with each object and evicts the one with lowest cost/size, Hybrid associates a utility with each object and evicts one with least utility to reduce total latency and others

4.5 Cache Coherency

Cache coherence mechanisms provide 2 types of consistency, Strong cache consistency and weak cache consistency

Strong cache consistency

Client validation – the proxy polls each time to check if the page has been modified.

Server invalidation – the server send out invalidate request for a modified page.

Weak Cache Consistency

Adaptive TTL – Adjusts the documents Time To Live based on observations of its life time

Piggyback Invalidation – Here the requests for validation or invalidation are piggybacked on proxy requests or client responses respectively.

Other methods for improving web caching schemes are user access predictions, load balancing, proxy placements, dynamic data caching, and understanding web traffic characteristics.

5 Parallelism

One way to improve performance in the network protocol subsystem is to exploit the availability of multiple processors in the host. Many approaches to parallelism in network protocols have been proposed. Here approaches have been classified on the basis of *unit of concurrency* or what it is that the processing elements do in parallel.

5.1 Layer Parallelism

In layer parallelism each protocol layer is a unit of concurrency. Specific layers are assigned to process specific elements and messages are passed between protocols through inter-process communication.
Advantage: Simple and defines a clean separation between protocol boundaries
Disadvantage: Concurrency is limited to number of layers in the stack and large amounts of context switching and synchronization between layers occur

5.2 Connection Level Parallelism

Connections form the unit of concurrency in connection-level parallelism. Speedup is achieved with multiple connections, which are processed concurrently.
Advantage: Simplicity and exploits the natural concurrency between connections
Disadvantage: No parallelism is achieved within a single connection

5.3 Functional Parallelism

A protocol layers functions are a unit of concurrency. Functions within a single protocol layer (e.g. checksum, ACK generation) are decomposed and assigned to a single processing element.
Advantage: Fine grained and can improve the latency of a single message as well as aggregate throughput

Disadvantage: Requires synchronization within a protocol layer and dependent on concurrency available between the functions in that layer.

5.4 Data level Parallelism

Data is the unit of concurrency. Processing units are assigned to the same functions but they all operate on different parts of the same message.
Advantage: Most fine-grained and the potential for the greatest improvement in throughput and latency.
Disadvantage: Processing elements must synchronize which may be expensive.

5.5 Packet Level Parallelism

Packets are the unit of concurrency
Advantage: Packets are processed regardless of which connection they are associated with or the layer in which they are present, achieving speedup for both single and multiple connections
Disadvantage: Requires locking shared state.

6 Bottlenecks

Here are some of the bottlenecks identified by the real-time application papers [13,14,15]:

- Main bottleneck is the inter-process communication cost while trying to provide a real-time performance.
- The context switching cost can also be a bottleneck if it is expensive since you often need to context switch between the user's software and operating system when communicating data to outgoing network.
- Memory access latency

7 Case Study: Distributed Video Streaming Over Internet

- Video streaming needs to face challenges of high bit rates, delay and loss sensitivity, hence TCP protocol is not suitable.
- This paper [14] provides a way to make multimedia content available at multiple sources so receiver can choose "best" sender based on BW, loss, and delay.

7.1 Receiver Side: Rate Allocation Algorithm

- Receiver is responsible for sending control information allocating the sending rates to all the senders based on estimated loss rate and available bandwidth.
- The allocation algorithm is run only a few times when there is a change in the estimated bandwidth.
- The algorithm to estimate bandwidth has high TLP since each of the sender's bandwidth can be determined independently.

- However, the use of estimated bandwidth information needs to be performed in one node to determine whether a change in the allocated sending rate should be made or not. This is most likely sequential in nature and its computational complexity will grow with the number of sources. The good news is that this step is done infrequently: only when estimated bandwidths have changed significantly.

7.2 Sender Side: Packet Partition Algorithm

- Each of the multiple senders keeps track of when to send a packet based on the last control information.
- A sender j only sends a packet k if k's arrival time at the receiver from source j is estimated to be largest.
- This estimation is consistent among all senders because it is solely based on the control packet received from the receiver.
- Each node needs to execute the algorithm to compute the estimated difference for all senders which also contains high amount of DLP (TLP) since each source's estimated difference can be computed independently.

8 Processor Architectures

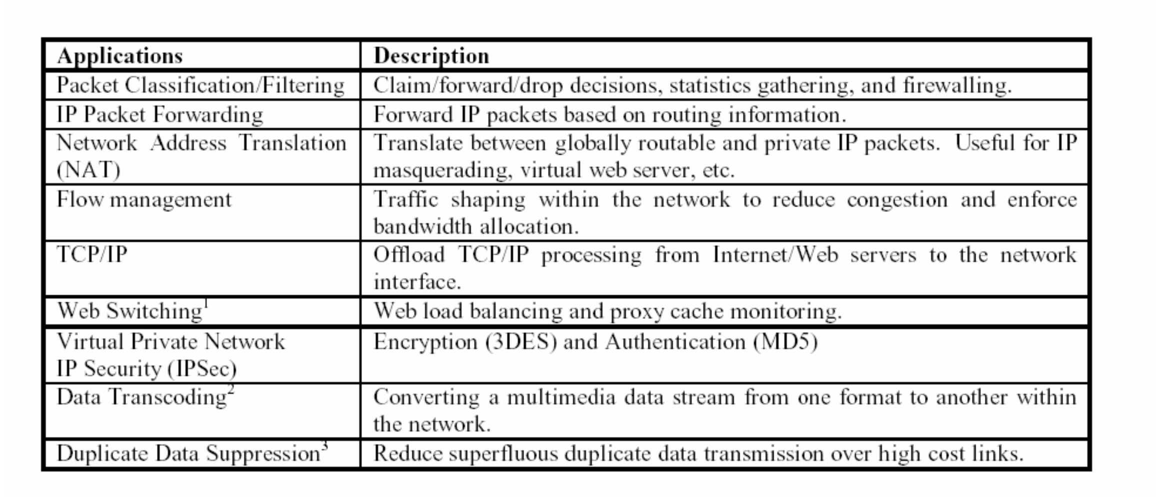
Rapid advancement in network technology has boosted the potential bandwidth of networks to the point that cabling is no longer the bottleneck. The bottleneck lies at the nodes of the network where data traffic is intercepted and forwarded. Specialized processors have been used at these nodes in order to handle data trafficking. These chips are called Networks Processors.

The role of the Network Interface (NI), which allows a computer system to exchange messages with other systems connected to the network, has grown in importance with the evolution of network technology. To meet the functionality and performance requirements of present and emerging network applications, the current trend is to use *programmable* microprocessors on *network interfaces* (PNI) that can be customized with domain-specific software.

To evaluate the various network architectures we need to consider what workloads must the processor support, what level of performance is required and what architecture delivers this level of performance.

8.1 Workloads to be supported by the network processor

The first six applications in Figure 11 possess a limited amount of data within the protocol headers of their packet and their processing requirements are independent of the packets overall size. But these applications maintain state tables in complex data structures that need to be searched on a per packet basis. The last three applications require significant processing capability to process packets at the network link rate.



Applications	Description
Packet Classification/Filtering	Claim/forward/drop decisions, statistics gathering, and firewalling.
IP Packet Forwarding	Forward IP packets based on routing information.
Network Address Translation (NAT)	Translate between globally routable and private IP packets. Useful for IP masquerading, virtual web server, etc.
Flow management	Traffic shaping within the network to reduce congestion and enforce bandwidth allocation.
TCP/IP	Offload TCP/IP processing from Internet/Web servers to the network interface.
Web Switching[1]	Web load balancing and proxy cache monitoring.
Virtual Private Network IP Security (IPSec)	Encryption (3DES) and Authentication (MD5)
Data Transcoding[2]	Converting a multimedia data stream from one format to another within the network.
Duplicate Data Suppression[3]	Reduce superfluous duplicate data transmission over high cost links.

Figure 11

8.2 Performance

The performance metric used is the number of messages per second that a given architecture can support for a given workload, which translates directly to network speeds that are enabled for the various architectures.

8.2.1 Details of the 4 network processors evaluated

Superscalar: A superscalar out of order processor with a 7 stage pipeline which uses a 7 stage pipeline, scoreboarding and register renaming to resolve dynamic dependencies. Instruction issue width

Fine-grained Multithreaded: Extends the core out-of-order superscalar processor with support for multiple hardware thread contexts

Chip-Multiprocessor: Partitions chip resources rigidly in the form of multiple processors. This architecture allows multiple threads to execute completely in parallel.
In this the issue width of each processor core is not scaled individually but increasing the number of single-issue cores scales the total processor issue width. Even though each core is under provisioned the speedups due to simultaneous issue from multiple cores is linear as seen in Figure 13 on the next page.

Simultaneous Multithreaded: Has hardware support for multiple thread contexts and extends instruction fetch and issue logic to allow instructions to be fetched and issued from multiple threads each cycle.

It can be seen from Figure 12 that the benchmarks exhibit limited ILP. Increasing the functional units beyond a certain point does not improve performance much. Also increasing other processor resources such as increasing the instruction queue length and renaming registers did not help much

The throughput for FGMT shows only a modest improvement (MD5) and remains unchanged for the others (3DES, ip4). Absence of long latency operations in these workloads does not give FGMT any significant advantage over SS.

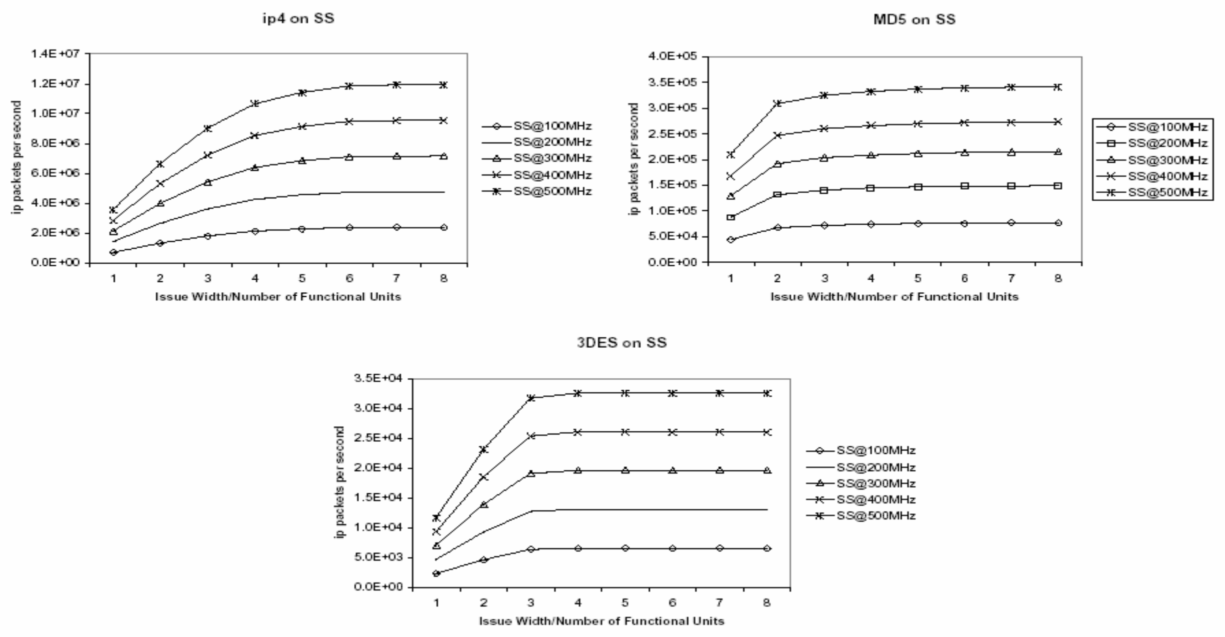


Figure 3. IP packet throughput achieved with each benchmark on a SS processor.

Figure 12

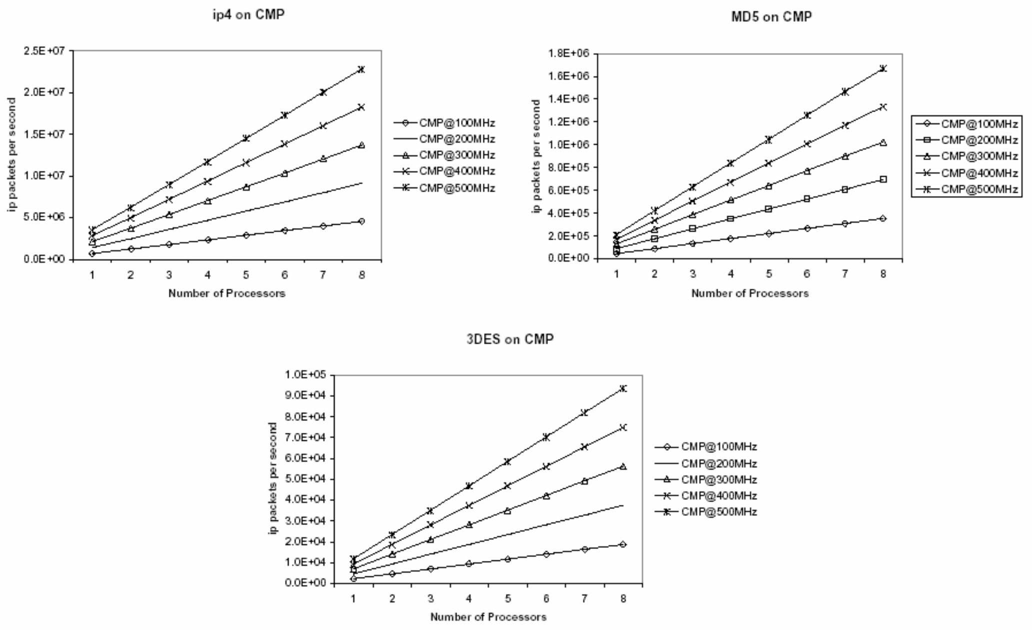


Figure 5. IP packet throughput achieved with each benchmark on a CMP.

Figure 13

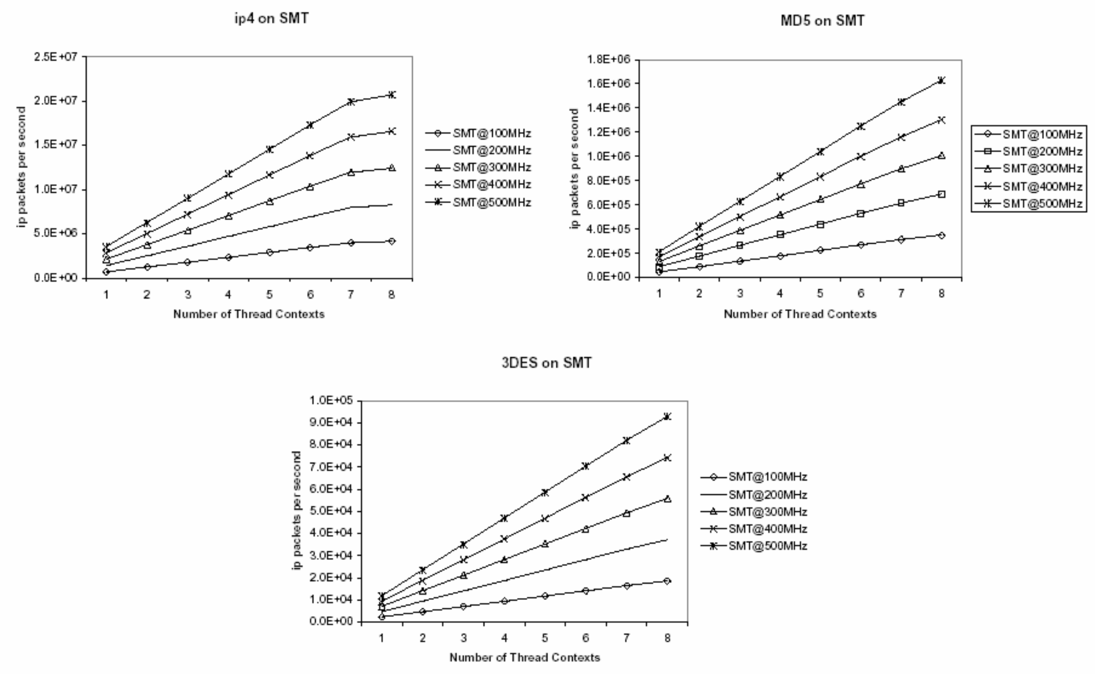


Figure 6. IP packet throughput achieved with each benchmark on an SMT processor.

Figure 14

SMT (Figure 14) combines the best features of previous architectures i.e. ILP and thread level parallelism. The performance of the benchmarks scale linearly with the number of contexts and issue-width.

8.2.2 Architectural Comparison

As can be seen in Figure 15, for both superscalar and FGMT architectures, the performance saturates with increasing number of contexts. For superscalar, this is because, as mentioned in previous section, there is a limited amount of ILP for superscalar architecture to exploit. For FGMT architecture, it is mainly useful for hiding long latency operations which does not happen often in networking applications. On the other hand, for CMP and SMT architectures, the performance scales linearly with the number of processors, contexts, and FUs that you have because their architectures can exploit TLP explicitly by executing multiple threads in parallel.

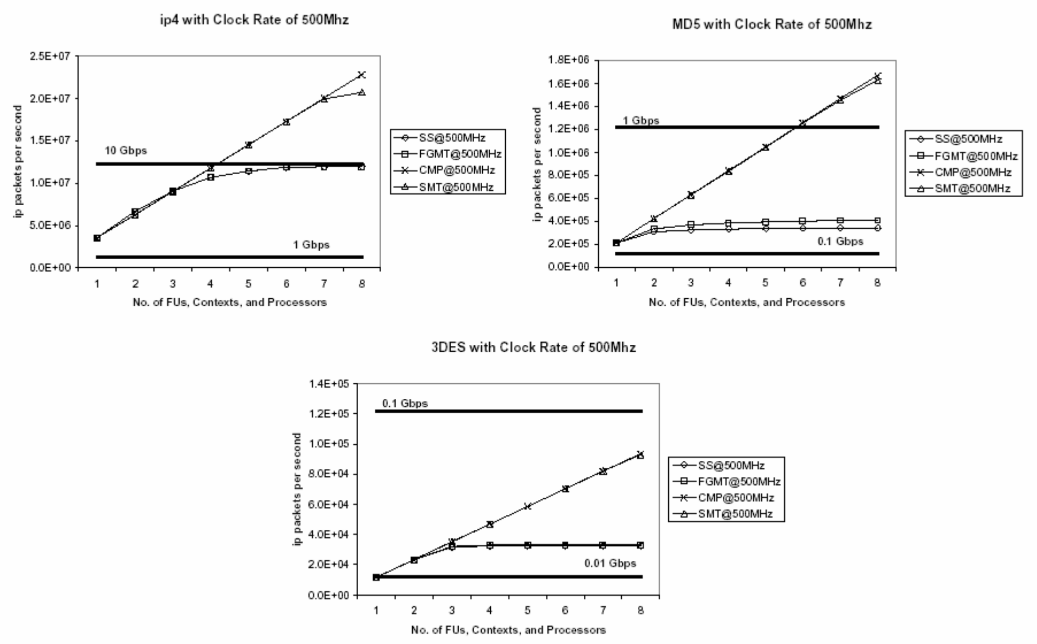


Figure 7. Performance results for all architectures, clocked at 500 MHz, running all benchmarks.

Figure 15

9. Benchmarks

With the increase of demand in networking application and the popularity of network processors, there is a need for specific benchmarks targeted for networking applications. The SPEC benchmarks which are used for general purpose processor are not suitable for network application because of the different characteristics of the applications, as mentioned above. As a result, there have been several benchmarks created, some targeted specifically for networking and other benchmarks targeted for general embedded processors, with a particular suite within it focused on networking applications. The most commonly used embedded benchmarks are the EEMBC suite, which divides the benchmark into 5 categories, with one of them being Networking. The networking benchmark within EEMBC included the following algorithm [6]:

- Dijkstra - algorithm to calculate the shortest path problem
- Patricia – data structure used in place of full trees, often used to represent routing tables in network application
- Packet flow

However, a problem with the EEMBC benchmarks is the fact that it is not readily accessible to academic because of its high cost[7] which resulted in the MiBench benchmarks from Univ. of Michigan. Similar to EEMBC, MiBench is partitioned into six different suite with each suite targeting a specific embedded application and one of the target is network. The network benchmarks within MiBench include the same Dijkstra and Patricia benchmark as well as a CRC32, cyclical redundancy check.

Other benchmarks in networking include Netbench and MediaBench from UCLA. Even though Mediabench targets media applications(multimedia and communication systems), it has several benchmarks which are suitable for network applications. However, because it lacks some of the other characteristics of network applications, the Netbench benchmarks were created to target network processors. The Netbench benchmarks are divided into 3 different type of applications [8]:

1) low/micro level – operations nearest to the links
2) routing level applications – IP level routing
3) application level programs – the most timing consuming which include parsing packet headers and making intelligent decision about packet destination

The Commbench from Washington University is another benchmark which focuses on streaming data flow based application as well as packet-based processing tasks such as routing and data forwarding [9]

10. Discussion

How do the different types of parallelism scale with data set?
How many threads are sufficient?

11. Conclusion

Network applications exhibit the following properties:

- Reasonable computation-to-memory operation ratio
- Predictable cache behavior
- Limited ILP but large amount of high level parallelism at packet level, data level and thread level
- Efficient mapping to CMP and SMT architectures

References

General Applications Characteristics

[1] http://cares.icsl.ucla.edu/pres.html

[2] Network Processing: Applications, Architectures and Examples ─ Tutorial at Micro 34, Dec. 2001

[3] Gokhan Memik and William H. Mangione-Smith. NEPAL: A Framework for Efficiently Structuring Applications for Network Processors. *Second Workshop on Network Processors – NP2 (held in conjunction with HPCA)*, Anaheim / CA, Feb. 2003

[4] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *International Symposium on Microarchitecture*, pp. 330–335, 1997.

Benchmarks

[5] Memik, G., W.H. Mangione-Smith, and W. Hu. NetBench: A Benchmarking Suite for Network Processors. In Proceedings of *International Conference on Computer-Aided Design (ICCAD),* pp. 39-42*,* Nov. 2001, San Jose / CA.

[6] http;//www.eembc.org

[7] Matthew R. Guthaus, et al , *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001.
[8] NetBench: A Benchmarking Suite for Network Processors Gokhan Memik, B. Mangione-Smith and W. Hu CARES Technical Report No. 2001_2_01

[9] T. Wolf and M. A. Franklin. CommBench : a telecommunications benchmark for network processors. In Proc. of IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, Apr. 2

Web Caching

[10] Jia Wang. A Survey of Web Caching Schemes for the Internet. Cornell Network Research Group (C/NRG) Department of Computer Science, Cornell University

[11] P. Rodriguez, C. Spanner, and E. W. Biersack, Web caching architectures: hierarchical and distributed caching, Proceedings of WCW'99.

[12] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, Exploring the bounds of Web latency reduction from caching and prefetching, Proceedings of the 1997 Usenix Symposium
on Internet Technologies and Systems, Monterey, CA, Dec. 1997.

Media Streaming

[13] Michael Schöttner, Andreas Kassler, Alfred Lupper, Piotr Dudzik, Peter Schulthess. Application Sharing – Architecture and Performance Aspects. Proceedings of the ACTS 2nd Mobile Communications Summit, Aalborg, Denmark, October 1997.

[14] T. P. Nguyen and A. Zakhor. Distributed Video Streaming Over Internet. *Proceedings of SPIE/ACM MMCN 2002*, January 2002.

[15] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. In *IEEE Conference on Distributed Computing and Systems*, July 2002.

Network Processors

[16] P. Crowley, M. Fiuczynski, J.-L. Baer, and B. Bershad. Characterizing processor architectures for programmable network interfaces. In *Proc. International Conference on Supercomputing*, Santa Fe, 2000.

[17] Erich M. Nahum, David J. Yates, James F. Kurose, and Don Towsley. Performance issues in parallelized network protocols. In *Proceedings of the First USENIX Symposium on OperatingSystems Design and Implementation (OSDI)*, pages 125–137, November 1994.