

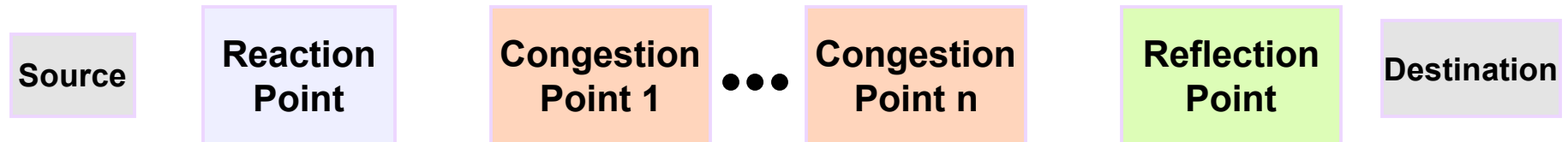
Some ideas for simple congestion management

**Balaji Prabhakar
IEEE 802 Plenary Meeting
Mar 14, 2007
Orlando, Florida**

Overview

- Congestion management scheme
 - Algorithm (what to feedback)
 - Signaling (how to feedback)
- Presentation format
 - Delineate solution space
 - Propose a scheme (algorithm and signaling)
 - Explore performance and complexity issues
- Further work and thoughts

Congestion management loop components



- Rough definitions...
- **Source:** Where data is generated.
- **Reaction Point:** Where the rate of injection of a flow (or flows) is changed due to congestion signals; usually, the place where rate limiters reside.
- **Congestion Point:** Where resources (buffers/links) exist and can be congested, and where congestion signals are generated; usually, switch buffers and the links they are attached to.
- **Reflection Point:** Where congestion signals are reflected back to the source.
- **Destination:** Where data is consumed.
- **Congestion Management Domain:** ReaP -- CPs -- RefP.
- Overarching goals: Good performance (high utilization, low delays, fair, good response, etc), low implementation complexity, low signaling overhead.

CM Architecture

- The overall architecture has the following components
 - Algorithm
 - What congestion signals should the CPs generate?
 - How should the ReaPs react to these signals?
 - Signaling
 - How should the congestion signals be sent back to the ReaP?
E.g. backward signaling or forward signaling
- Types of algorithm
 - Congestion management
 - Rate allocation
- Types of signaling architecture architecture
 - 3-point architecture: ReaP --> CPs --> RefP (forward signaling)
 - 2-point architecture: ReaP --> CP/RefP (backward signaling)

Rate allocation vs congestion management

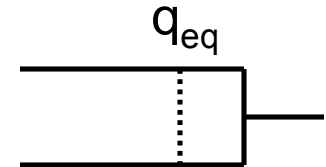
- Rate allocation, especially max-min rate allocation, is definitely a solution to the congestion management problem. But it “over-solves” the congestion management problem.
 - Notably, max-min allocation even at a single node is equivalent to a processor-sharing service discipline. Therefore, it entails “per-flow” work. In the network context, the per-flow queues are moved to the edge in the form of rate limiters.
 - Congestion management is, therefore, a smaller problem than rate allocation. This gives the hope that it is simpler.
- Secondly, a rate allocation algorithm needs to know the capacity of the link which it is allocating. When this changes in an unknown fashion, rate allocation involves two steps: determine the link capacity, then allocate it.
 - Congestion management copes with changing link capacities by simply “dinging” the one or two biggest flows. It gets away with this because it does not have equal rate allocation as a goal.
- Fairness is achieved by congestion management schemes as follows
 - Short-term, not packet-by-packet.
 - Proportional fairness, not max-min fairness.
 - Fair congestion management schemes favor the growth of low rate flows and “ding” high rate flows when there is congestion.

CM Algorithm

- We now focus on the algorithm and come to signaling later
- Since the algorithm only involves mechanisms at the ReaP and at the CP (the RefP is not involved in the algorithmics), we describe the ReaP and CP mechanisms
- Packet Format
 - We use a Congestion Indication field in the packet header, which we assume will be 6 or 7 bits long. This field will tell the ReaP the amount of congestion in the network. The variable that measures congestion is indicated “Fb.”
 - We will see that Fb values can only be 0 or negative, not positive.
 - As an *option*, we could also have a FlowID field in the header.
 - Every packet leaving a ReaP will have Fb value equals 0.

Congestion Point Mechanism

- At the CP
 - Sample packets with probability p
 - Compute: $Fb = - [q_{off} + w q_{delta}]$
 - If $Fb < 0$, and if Fb value is smaller (more negative) than Fb value in the packet header, then overwrite Fb value in packet header with computed Fb value
 - Else, do nothing
 - Note: w is a parameter, usually a power of 2



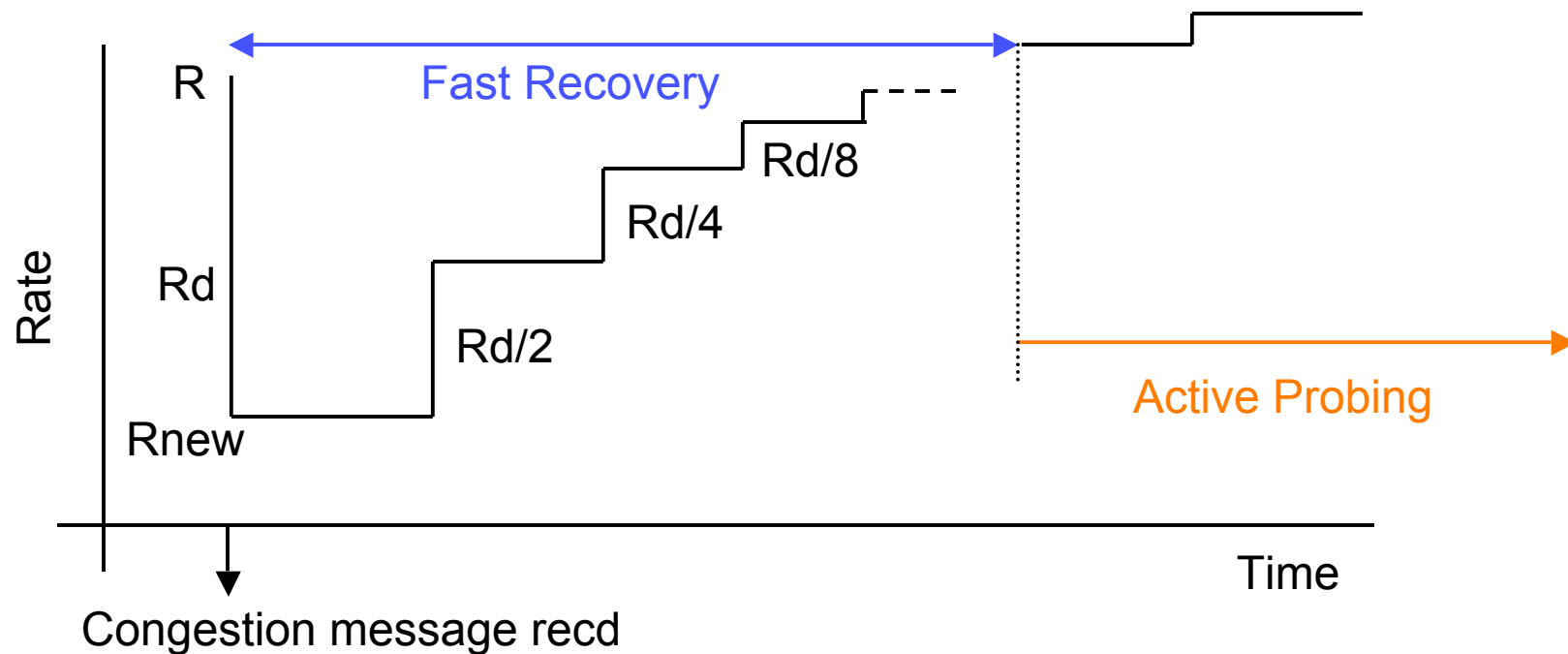
$$q_{off} = q - q_{eq}$$
$$q_{delta} = \# \text{ pkts enqueued} - \# \text{ pkts dequeued}$$

between two packet arrivals (or sampling instants)

Reaction Point

- ReaP Dynamics
 - Starting rate for every flow equals 10Gbps
 - Insert $Fb = 0$ in outgoing packet
 - Insert a “flowid” into outgoing packet’s header (optional)
- Whenever a ReaP receives a message
 - Decrease rate from R to $R_{new} = R(1 - IFb \times Gd)$, where Gd is a gain
 - Perform “fast recovery” and “active probing”
- Fast recovery
 - Let $Rd = R IFb \times Gd$ be the amount of rate decrease; the idea of fast recovery is to quickly regain as much of this rate as possible
 - Fast recovery proceeds in cycles; each cycle clocked by the “fast recovery timer”
 - Fast recovery timer
 - Increments for every transmitted byte up to B_{FR} , it then resets to zero and counts again
 - The cycles of counting are numbered 0, 1, 2, ...
 - The transmission rate is as follows
 - During cycle 0, at rate R_{new}
 - During cycle 1, at rate $R_{new} + Rd/2$
 - During cycle 2, at rate $R_{new} + Rd/2 + Rd/4$
 - And so on until cycle 5, as long as it does not receive any further QCN messages
 - If a congestion message is received, then cut rate as before and restart fast recovery
 - At the end of fast recovery, the source moves to Active Probing

Reaction Point



- Active probing (multiplicative increase) always follows fast recovery
 - Active probing is clocked by the “rate increase timer”
 - Rate increase timer expires every T secs
 - When timer expires, the current rate is changed to $R \cdot A$, where $A > 1$ is the increase parameter
 - If a congestion message is received, cut rate as before, perform fast recovery and then active probing

Related work and some notes

- The algorithm suggested is heavily influenced by several algorithms
 - It calculates F_b as in BCN, and the REM and PI controllers in the Internet literature
 - The fast recovery part is from the BIC TCP algorithm
- The CP computes $F_b = - [q_{off} + w q_{delta}]$
 - This is different from BCN but similar to REM and PI. It is based on the observation that even in BCN the sources merely use F_b and not q_{off} or q_{delta}
 - More fundamentally, F_b becomes the “congestion measure;” a part of it depends on buffer occupancy, another part depends on link utilization, and the switch is best placed to decide how to weigh these resources.
- Simplifications with respect to BCN come mainly from quantization, and lack of CP--RP association through the CM tag.
- A major benefit: Not having the CM tag on allows for perfectly incremental deployment; i.e. no flowid, no CPID, nothing. Congestion messages use SA of sampled packet.
 - Moreover, in this case, we only need exactly one rate limiter per ReaP.

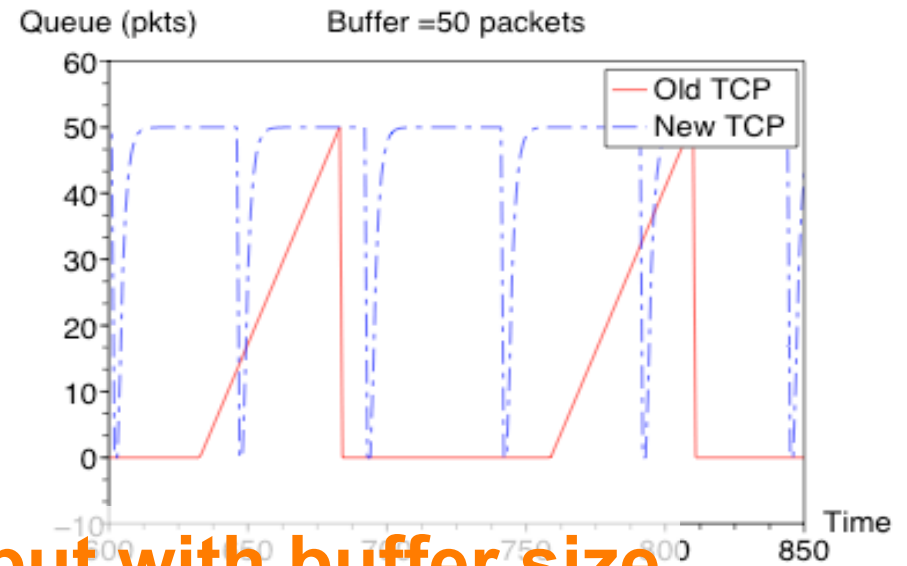
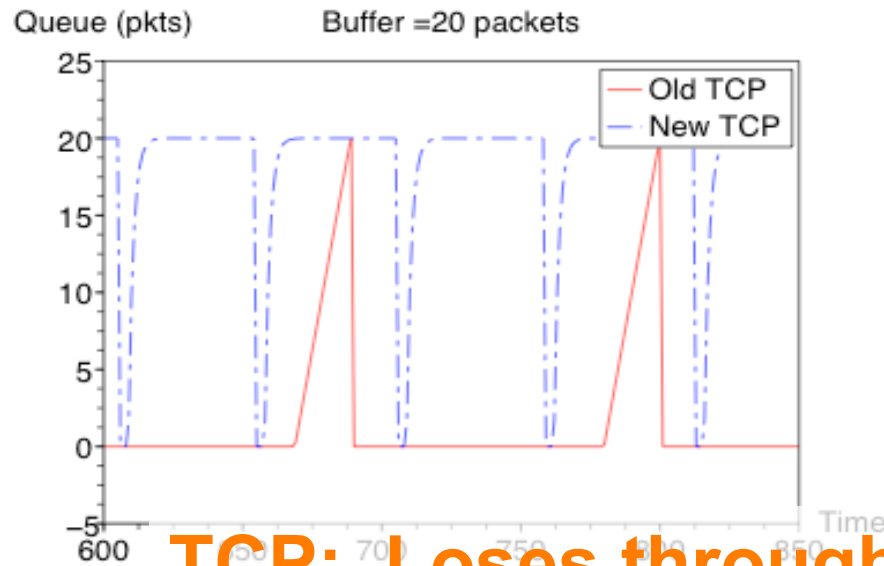
Intuition for Fast Recovery

- The best way to understand fast recovery is by considering TCP
 - Recall
 - TCP cuts the window by 1/2 for every dropped/marked packet
 - And increases window size by 1 each round trip time when there is no drop
 - When the bandwidth-delay product is high, the additive increase by 1 can be very slow, require large buffers, and lead to poor link utilization.
 - Fast recovery, or BIC TCP, is a method for overcoming these problems.
- BIC TCP through an example
 - Suppose the congestion window size equals 512
 - If we now get a packet drop, the window goes down to 256
 - After 1 RTT, if there are no drops, the window increases to $256 + 128 = 384$
 - After another RTT (if there are no drops), the window increases to $384 + 64 = 448$
 - The complete sequence of window sizes is:
 - 512, 256, 384, 448, 480, 496, 504, 508, 510, 511
- Active (additive) probing would start now, increasing the window size by 1, so we get window sizes: 512, 513, 514, ...
- If another drop occurs any time during this process, we go back to binary increase

Benefits of BIC TCP

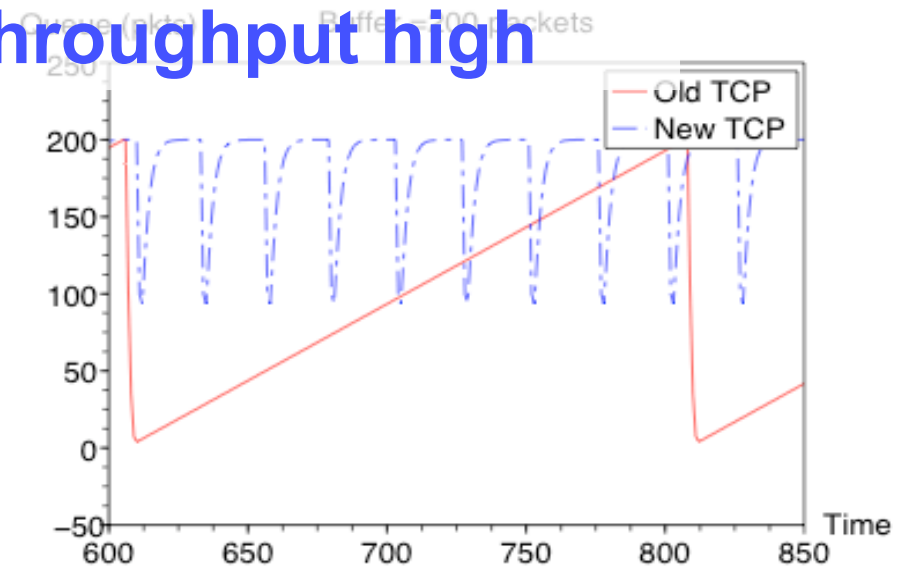
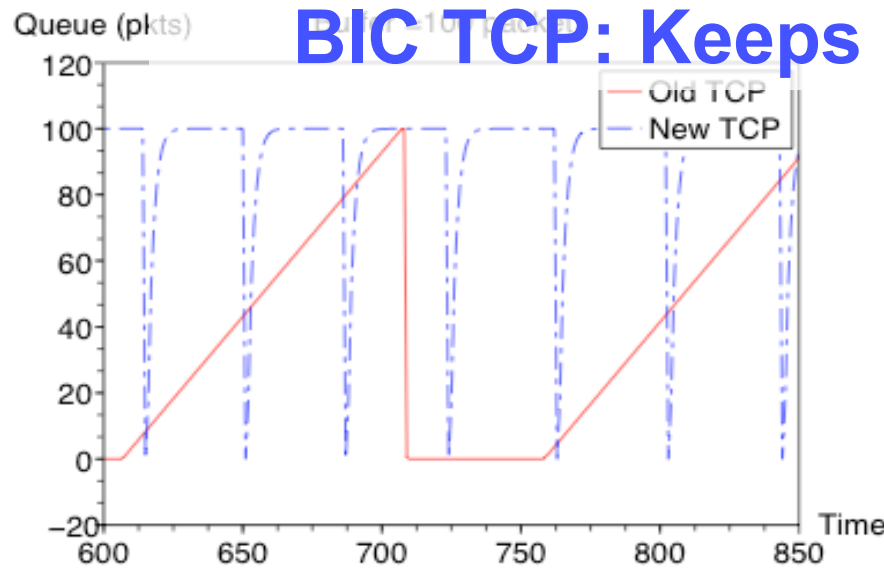
- BIC TCP
 - Gets sources high throughputs
 - Keeps links highly utilized (this is not exactly the same as the previous point)
 - *Most importantly*, it does this with small buffers
 - By doing a binary search for the correct window size, *instead of* a linear search, BIC TCP is both quick and gentle in probing for extra bandwidth
 - **Note: BIC has been invented by Rhee et al**
- Comparison of BIC with plain old TCP
 - Consider a link with capacity 1000 pkts/sec
 - A RTT of 200 msec
 - Bandwidth-delay product worth of buffering = 200 pkts
 - TCP will not give 100% throughput with less than this amount of buffering for a single source
 - We use buffers of depth 20, 50, 100 and 200 pkts

BIC TCP: Queue sizes

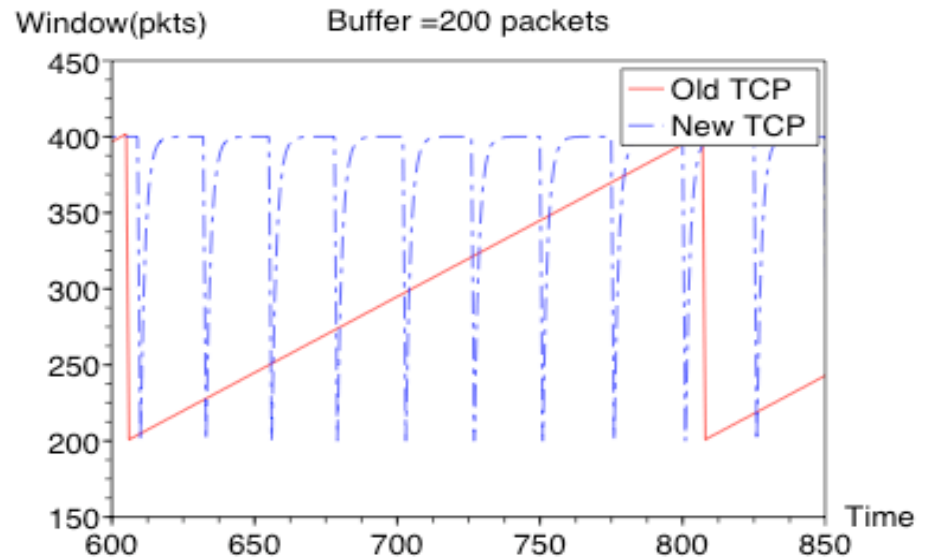
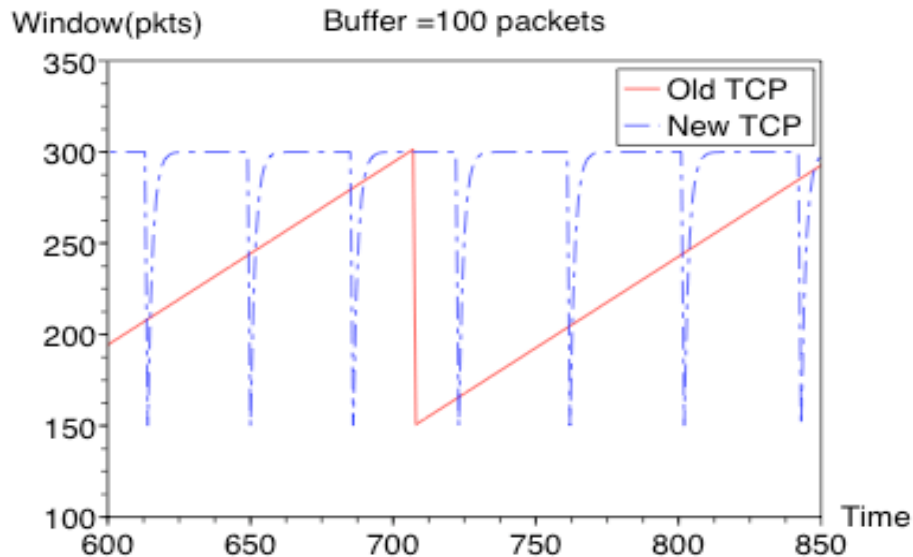
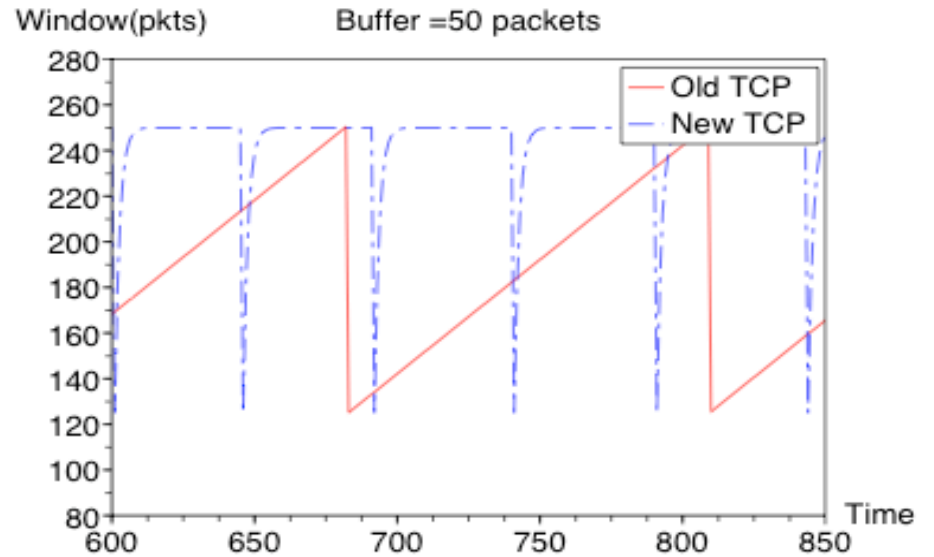
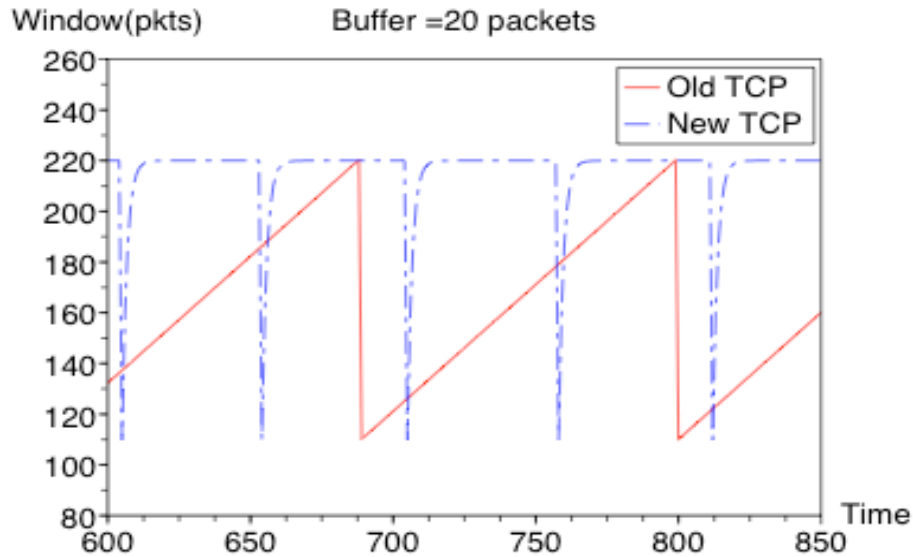


TCP: Loses throughput with buffer size

BIC TCP: Keeps throughput high



BIC TCP: Window sizes



Signaling and Reflection Point Dynamics

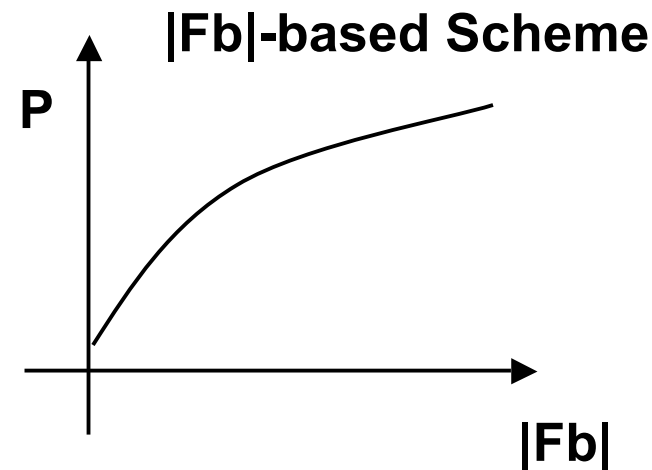
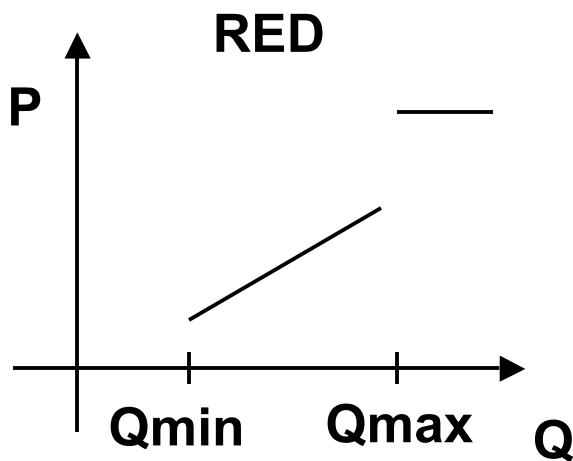
- Signaling, some motivation: Consider backward signaling as in BCN
 - Amount of signaling
 - BCN uses a sampling probability of p (= 1%, with extra when congestion increases)
 - Without the congestion-related enhancement, the amount of signals is proportional to the *amount of traffic* and not to the *amount of congestion*
 - Node- vs path-centric signaling
 - BCN generates signals from each congested node, doing it per path will again reduce the amount of signaling
- So, if forward signaling can fix these problems (the first can be fixed by BCN itself), why not jump all the way to the Internet schemes
 - This means just 1-bit per packet (Fb encoded in 1 bit)
 - And will double the number of packets if we have per-packet acks.
 - If we gather up multiple acks per source (or flow) at the RefP and reflect that...
 - We have to do per-source (or per-flow) work.
 - It is actually per-RTT amount of work or per-flow, whichever is minimum.

Signaling

- There is also more going on in the Internet...
 - Packet sequence numbers, RTT estimation, etc
 - So, there is more state at each source which is usable, the 1-bit is misleading
- So while a 1-bit scheme is really tantalizing, esp because we could use the DE (Discard Eligible) bit, it imposes an overhead.
- Actually, the Ethernet congestion management problem is interesting because
 - Ethernet is not a state-ful network like ATM.
 - Ethernet hosts do not keep transport-related state like in TCP/IP.
 - This really forces a fundamental rethinking of algorithm and signaling.
- To conclude, we want signaling to
 - Depend on amount of congestion and be path-centric.
 - Not be complicated to implement.
- We propose a multi-bit, congestion-dependent signaling scheme

Forward signaling and RefP Dynamics

- Each CP computes Fb per packet
 - It replaces the Fb value in the packet with the computed value if the computed Fb is more negative
 - Thus, Fb values can only decrease as a packet goes through its path
- RefP Dynamics
 - When a packet is received, reflect the Fb value to the source with a probability which increases with |Fb|



To summarize

- The scheme we propose has
 - Reaction Points, Contestion Points and Reflection Points.
 - 1. Reaction Points: Insert $Fb = 0$ in outgoing packets. When congestion message arrives: perform multiplicative decrease, fast recovery and active probing.
 - 2. Congestion Points: Compute Fb , overwrite Fb in packet header.
 - 3. Reflection Points: Reflect Fb values to ReaPs with a probability biased by the Fb value in the packet.

Further work, refinements

- Due to packet-level, random effects some refinements of the basic algorithm are useful to make
 - The main point is that Control Theoretic analyses lose their fidelity with the discrete, packet-level, random effects present in the real network, *especially* when the buffers are short. This needs us to be careful when going from theory to practice. The following points are worth noting in this regard.
 1. BIC TCP makes a binary fast recovery, as opposed to linear or constant recovery. Binary and linear recovery delineate extreme points in a spectrum from more aggressive to less aggressive recovery. We could, of course, use something in between.
 2. Suppose a source gets multiple congestion messages in a burst, driving its rate down by a lot. Say that the rates of decrease were $Rd1$, $Rd2$ and $Rd3$. Fast recovery only uses the *last* amount of decrease $Rd3$. Using $Rd1+Rd2+Rd3$ or $\max(Rd1, Rd2, Rd3)$ for fast recovery, *when* congestion messages arrive in a burst, improves performance.
- A significant advantage of forward signaling is that *both decrease and increase* messages can be signaled by the reflection point, without the need for CM-type tags. This is purely because forward signaling gathers “path capacity,” not node capacity. This feature is not explored in our current proposal, but is worth considering.
- Another aspect has to do with the byte-counter timers. These timers make the system self-clocked and are hence v.useful. However, when the source transmission is very low, the timers can take too long to expire. So some “state recovery timers” may be needed here.