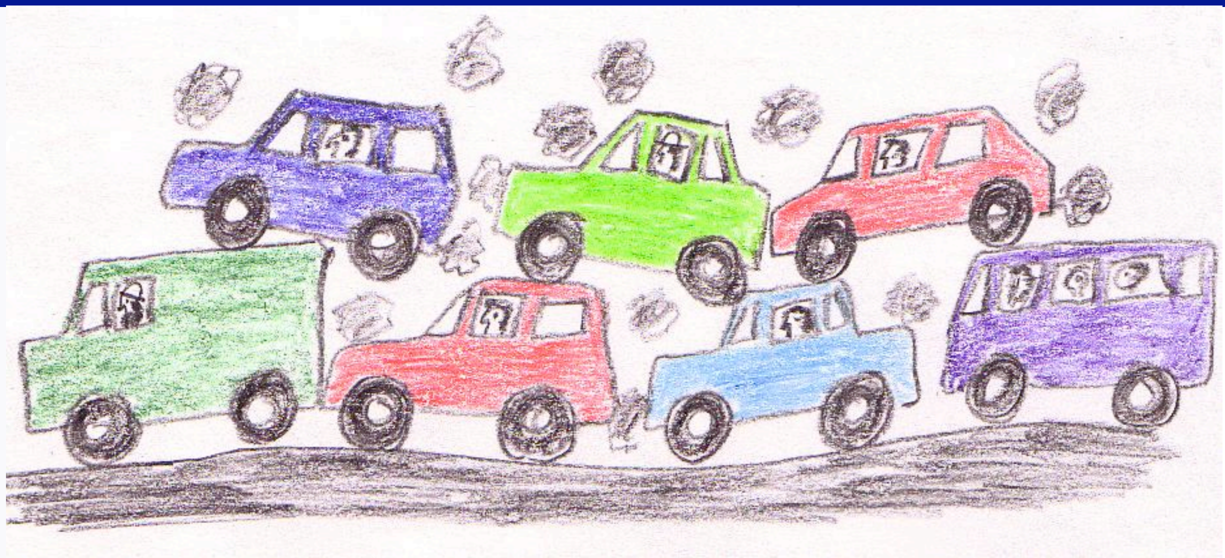
Congestion Control

Mark Handley



Outline

Part 1: “Traditional” congestion control for bulk transfer.

Part 2: Congestion control for realtime traffic.

Part 3: High-speed congestion control.

Part 1

"Traditional" congestion control for bulk transfer.

Congestion Control

End-to-end congestion control serves several purposes:

- Divides bandwidth between network flows in a "reasonably fair" manner without requiring per-flow scheduling by routers.
- Prevents congestion collapse of the network by matching demand to supply to ensure overall goodput remains reasonably high.

Congestion Collapse

Congestion collapse occurs when the network is increasingly busy, but little useful work is getting done.

Problem: *Classical congestion collapse*:

Paths clogged with unnecessarily-retransmitted packets [Nagle 84].

Fix:

Modern TCP retransmit timer and congestion control algorithms [Jacobson 88].

Fragmentation-based congestion collapse

Problem:

Paths clogged with fragments of packets invalidated because another fragment (or cell) has been discarded along the path. [Kent and Mogul, 1987]

Fix:

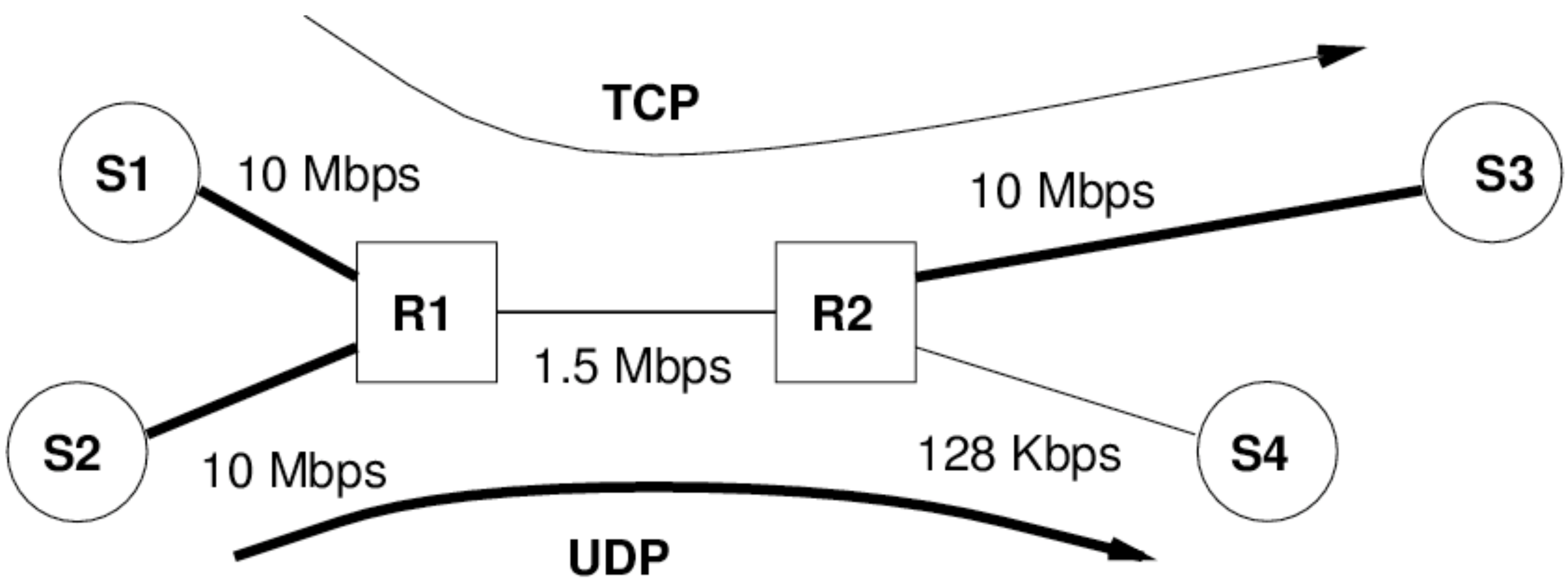
MTU discovery [Mogul and Deering, 1990]

Early Packet Discard in ATM networks [Romanow and Floyd, 1995].

Congestion collapse from undelivered packets

Problem: Paths clogged with packets that are discarded before they reach the receiver [Floyd and Fall, 1999].

Fix: Either end-to-end congestion control, or a ``virtual-circuit'' style of guarantee that packets that enter the network will be delivered to the receiver.

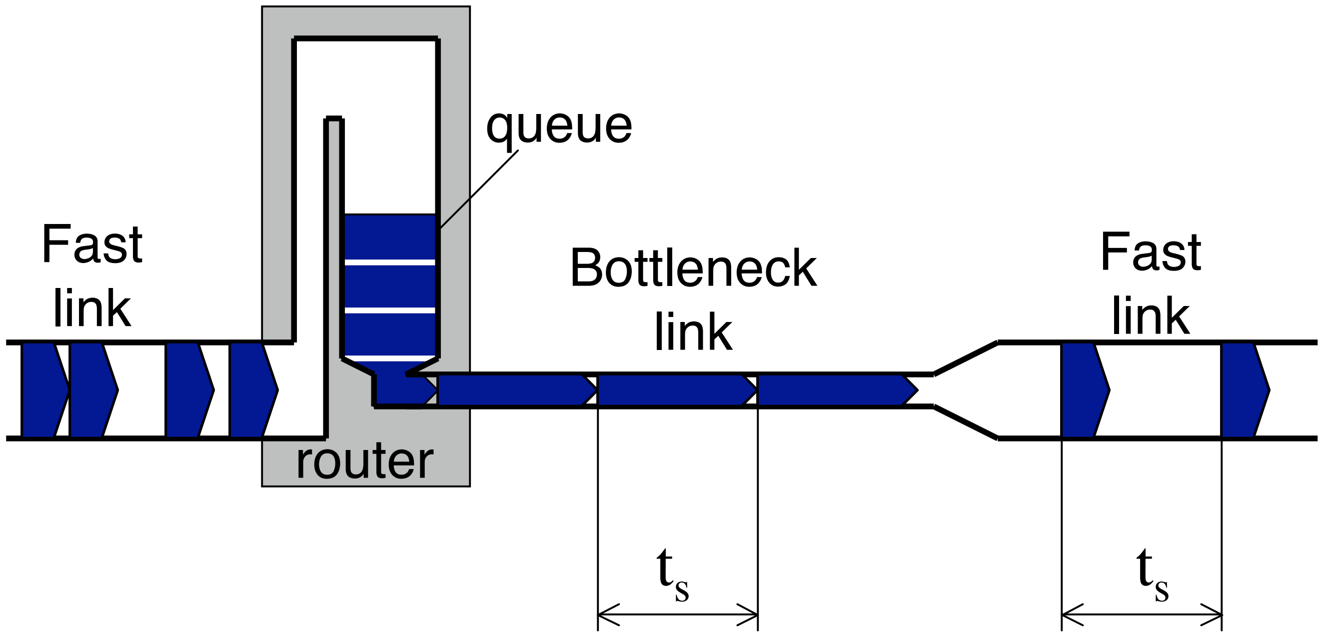


Congestion Control

Since 1988, the Internet has remained functional despite exponential growth, routers that are sometimes buggy or misconfigured, rapidly changing applications and usage patterns, and flash crowds.

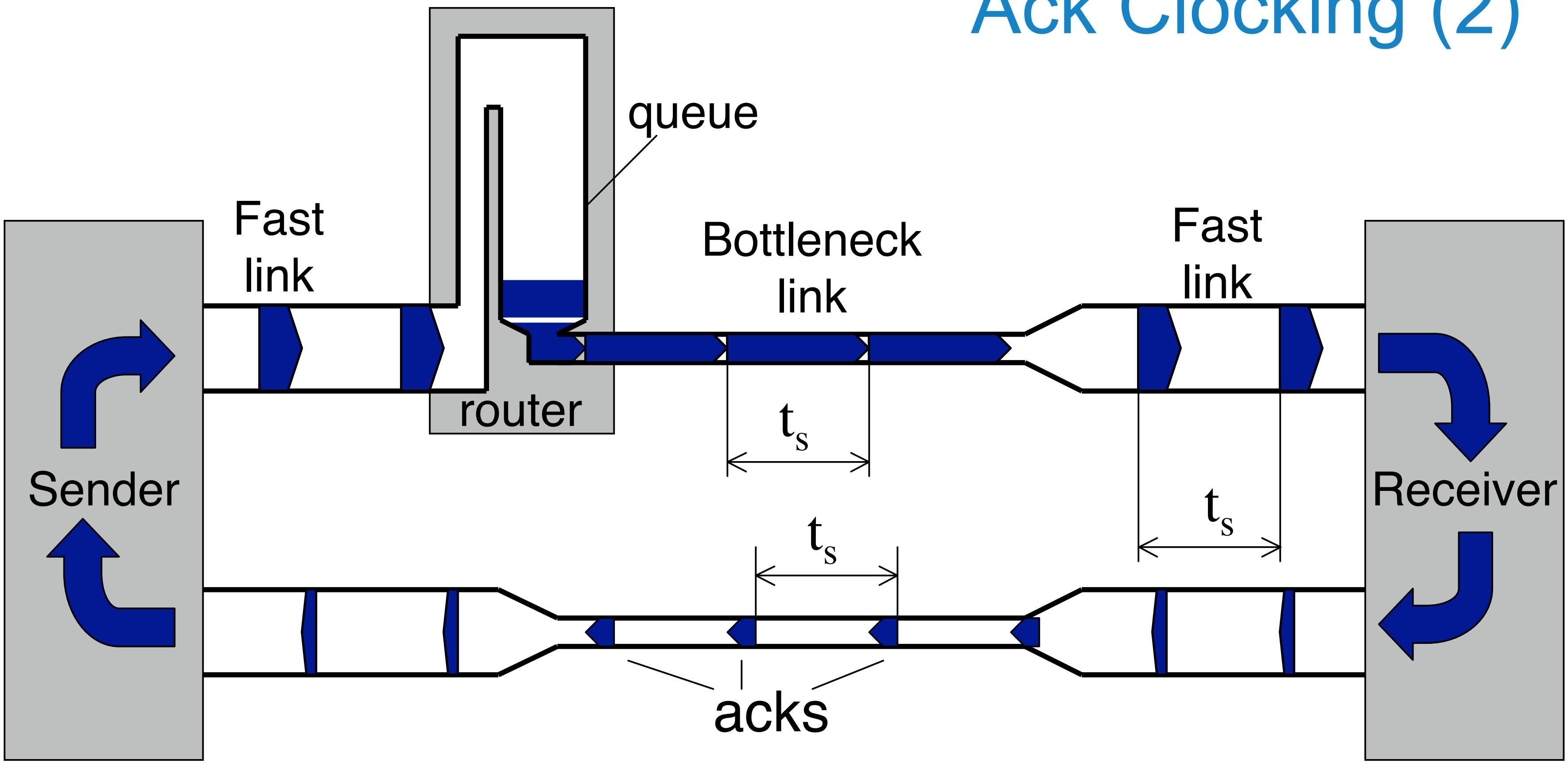
> This is largely because most applications use TCP, and TCP implements *end-to-end congestion control.*

Ack Clocking



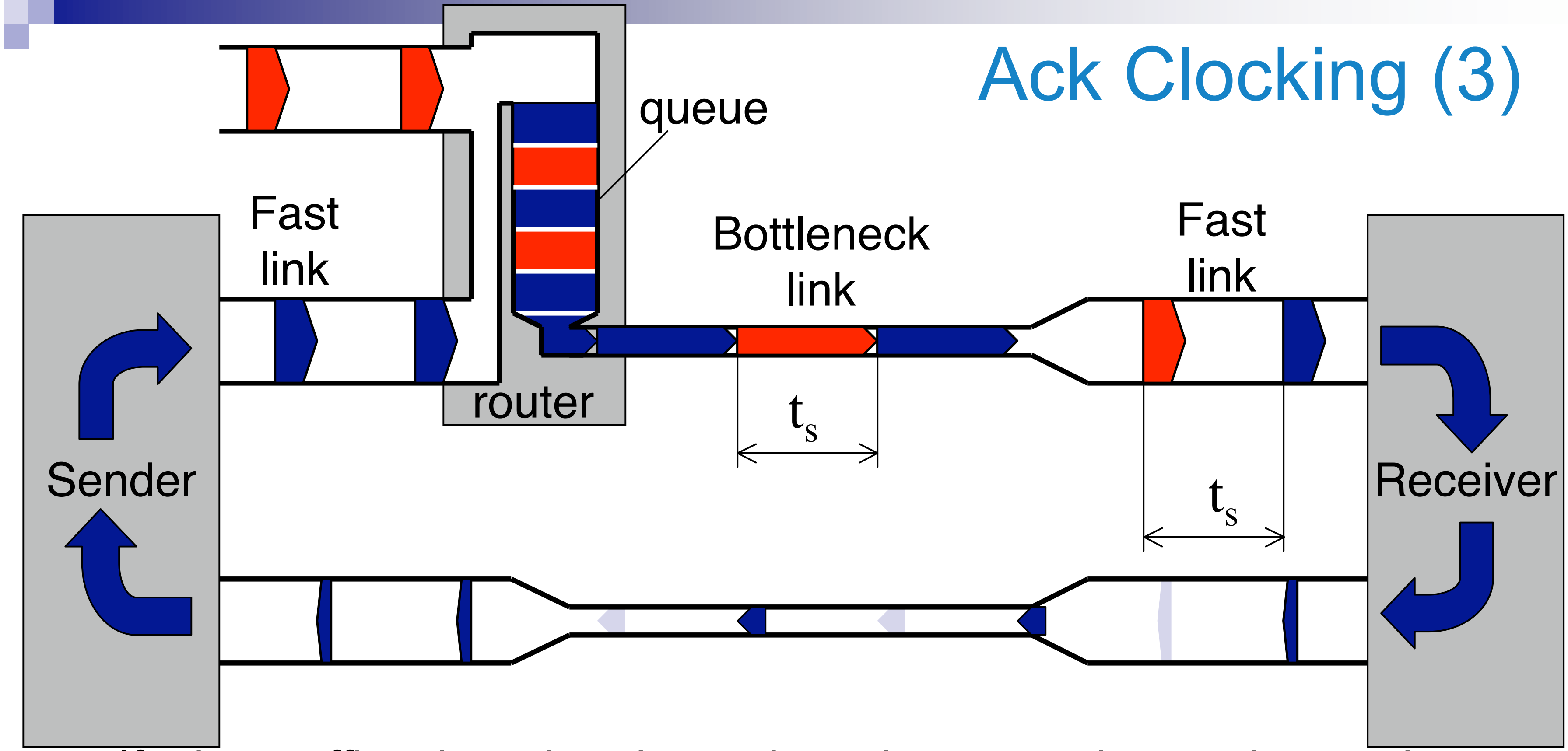
- A bottleneck link will space packets out in time, according it its service rate.
- The inter-packet spacing is preserved when packets leave the link (although later queuing can disturb it if there is cross traffic)

Ack Clocking (2)



- Receiver acks immediately and sender sends only when an ack arrives.
- Result: sender sends at *precisely* the rate to keep the bottleneck link full

Ack Clocking (3)



- If other traffic mixes, it reduces the ack rate, so the sender sends more slowly without changing its window.
- This automatic slowdown is important for stability. More packets end up in the queue, but they still enter at the same rate they depart.
- If there’s not space, a drop occurs, TCP halves its window, and the queue decreases.

Congestion Window

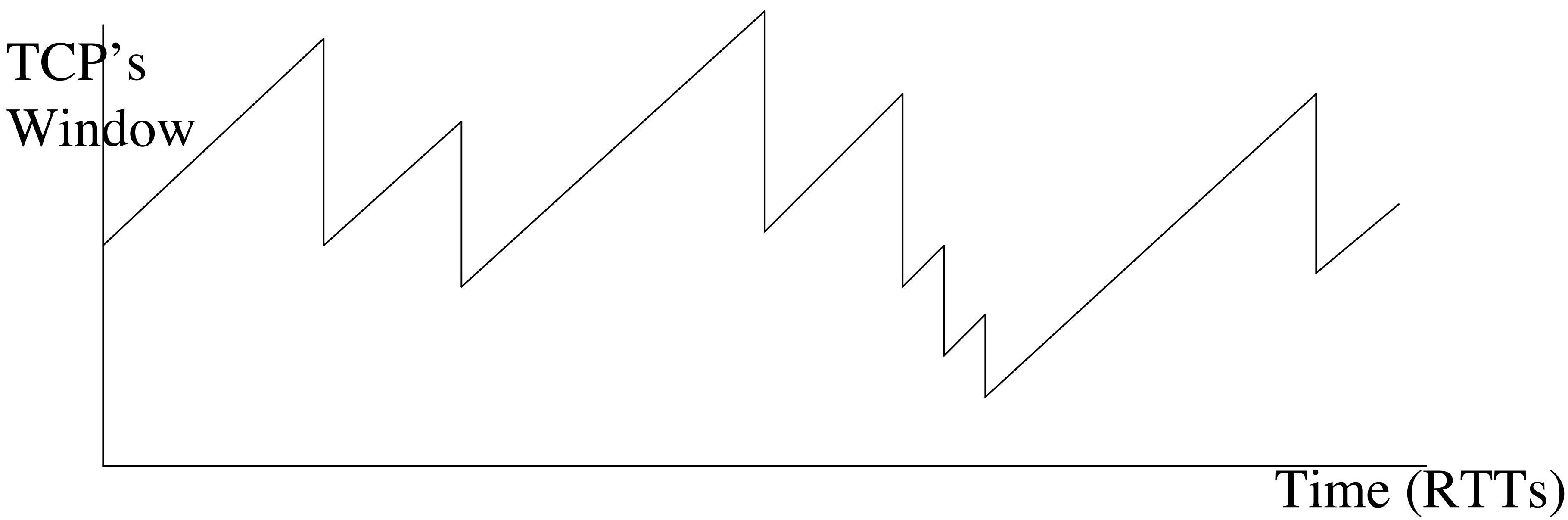
- So ack-clocking of a window of packets has nice stability properties.
  - It's harder to control the rate and get these same properties.
  - Rate control gives no automatic backoff if other traffic enters the network.

- The key question then is how large a window to use?

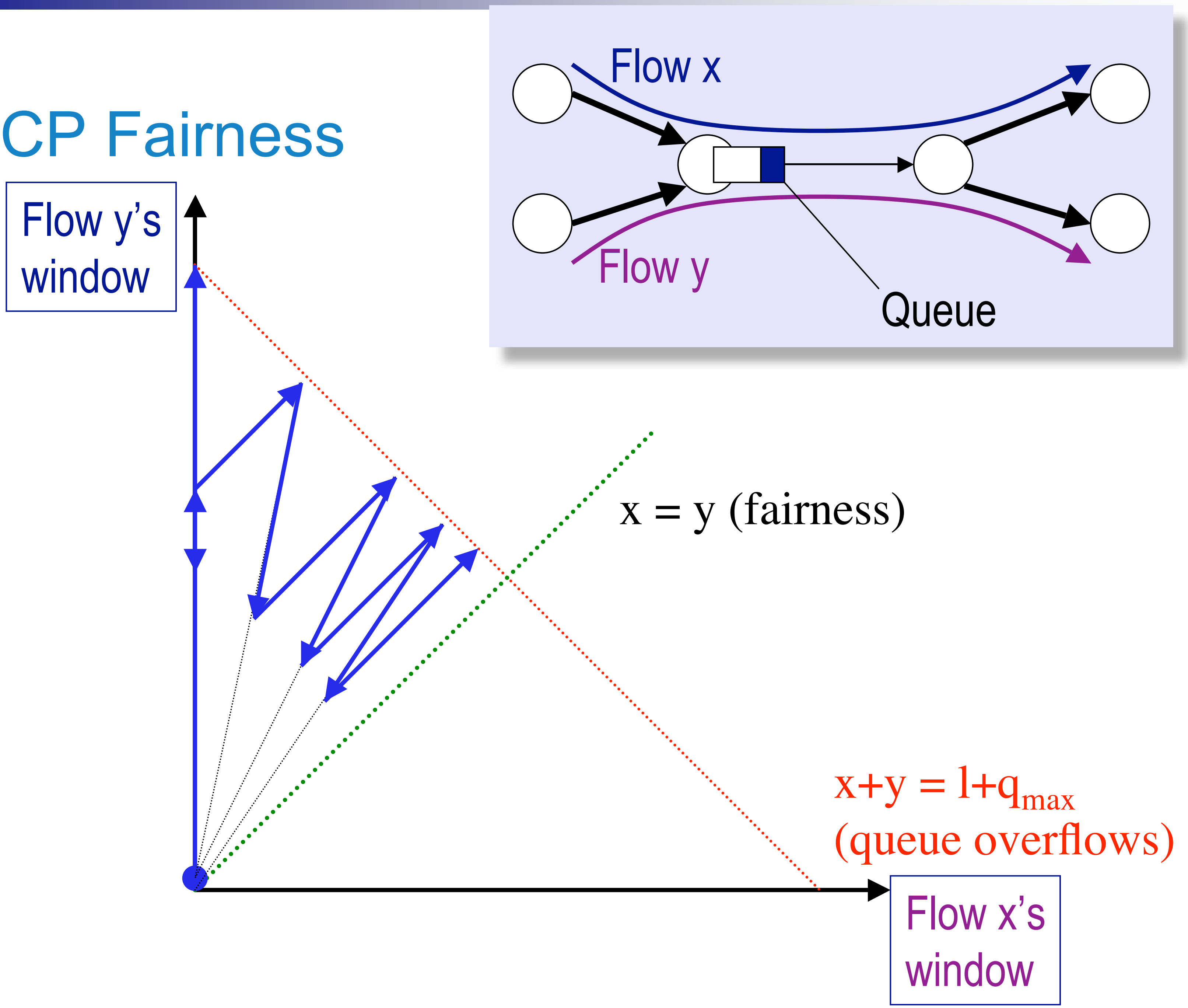
TCP Congestion Control

Basic behaviour: Additive Increase, Multiplicative Decrease (AIMD).

- Maintain a *window* of the packets in flight:
  - Each round-trip time, increase that window by one packet.
  - If a packet is lost, halve the window.



TCP Fairness



TCP (Details)

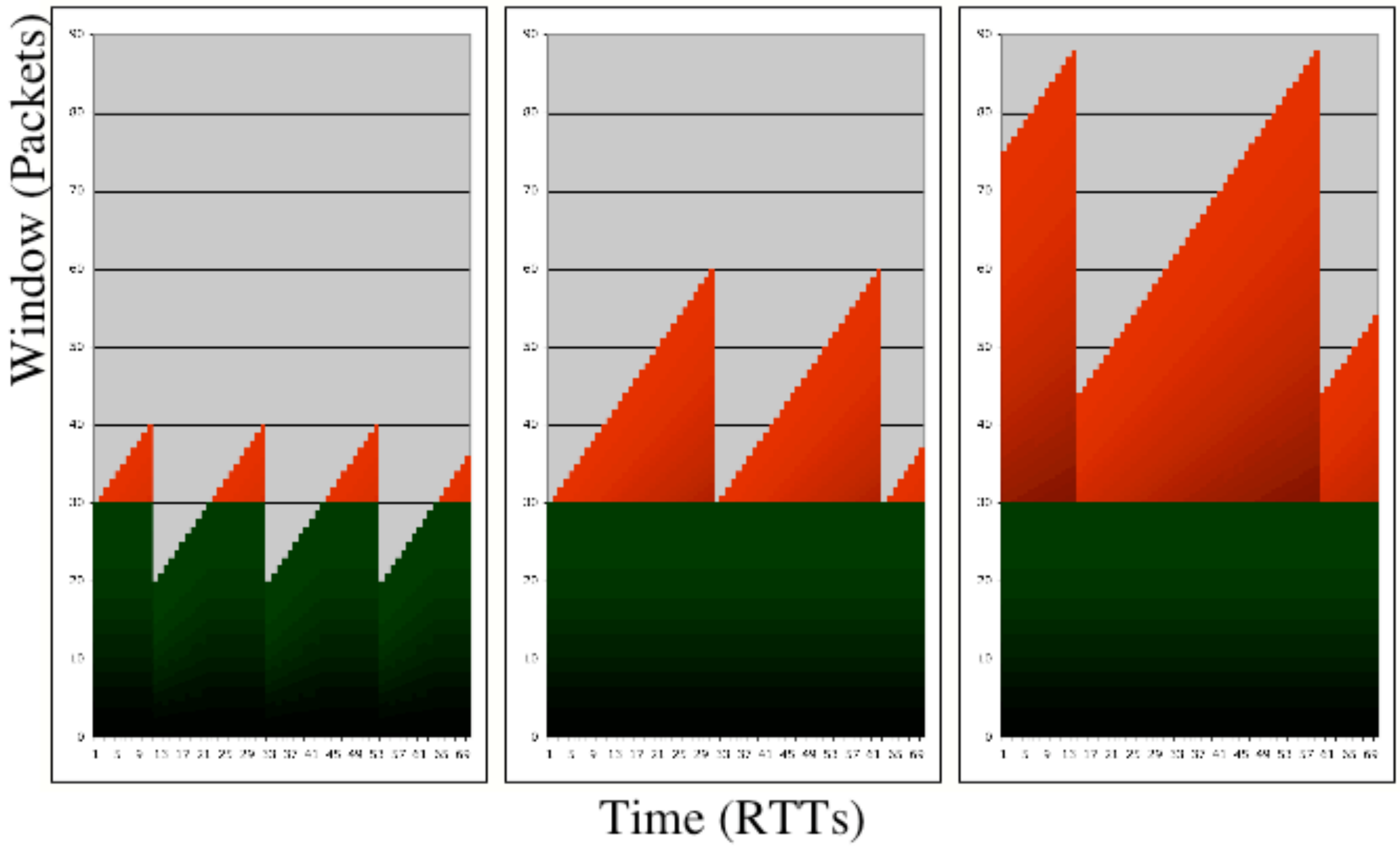
- TCP congestion control uses AIMD:
  - Increase the congestion window by one packet every round-trip time (RTT) that no packet is lost.
  - Decrease the congestion window by half every RTT that a packet loss occurs.
- In heavy congestion, when a retransmitted packet is itself dropped or when there aren't enough packets to run an ACK-clock, use a retransmit timer, which is exponential backed off if repeated losses occur.
- Slow-start: start by doubling the congestion window every roundtrip time.

Queuing

- The primary purpose of a queue in an IP router is to smooth out bursty arrivals, so that the network utilization can be high.

- But queues add delay and cause jitter.
  - Delay is the enemy of real-time network traffic.
  - Jitter is turned into delay at the receiver's playout buffer.
  - Understanding and controlling network queues is key to getting good performance from networked multimedia.

TCP Throughput and Queue Size

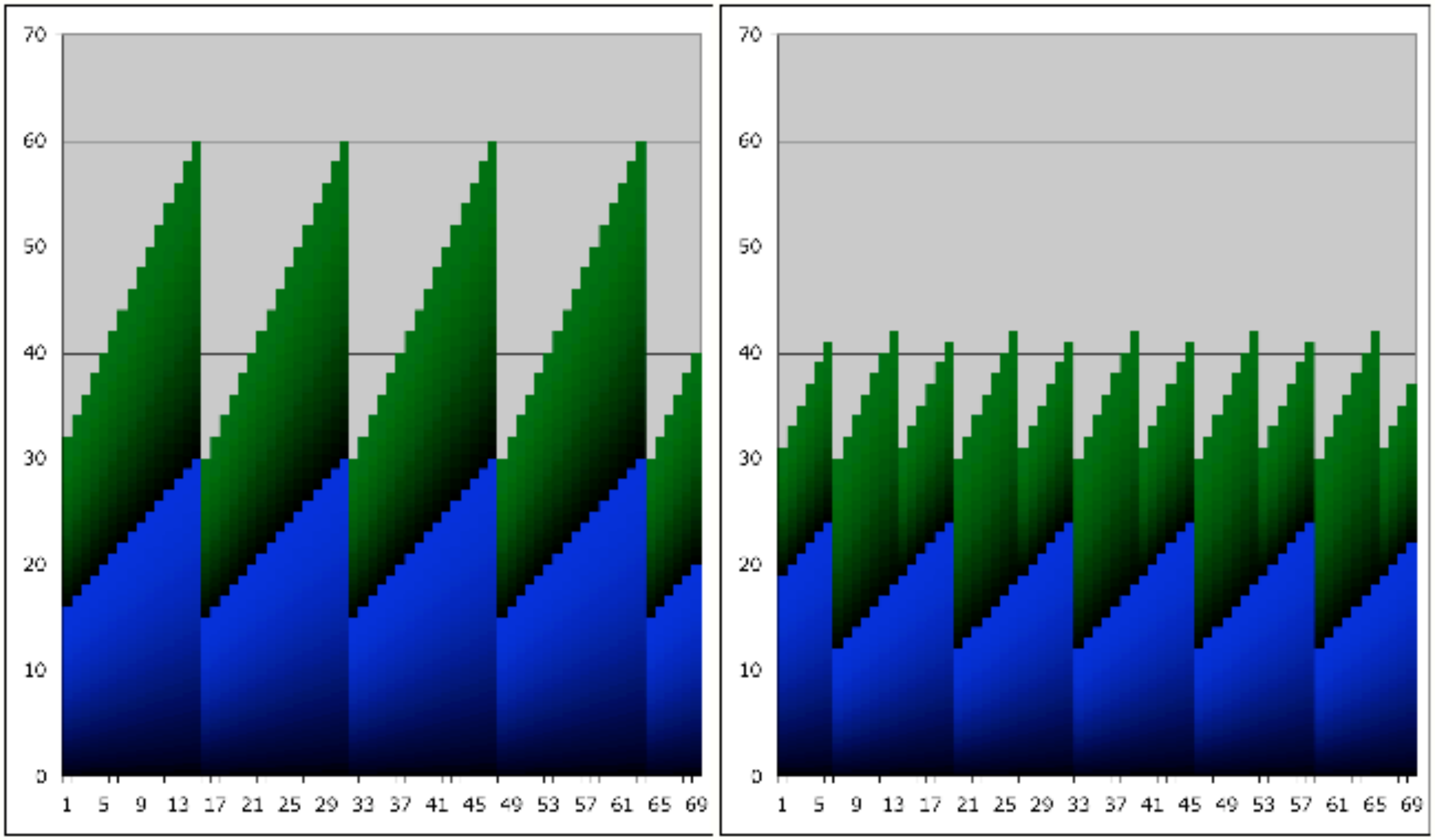


Green: packets in transit. Red: packets in the bottleneck queue

TCP and Queues

- TCP needs one delay-bandwidth product of buffer space at the bottleneck link for a TCP flow to fill the link and achieve 100% utilization.
- Thus, when everything is configured correctly, the peak delay is twice the underlying network delay.
  - Links are often overbuffered, because the actual RTT is unknown to the link operator.
  - Real-time applications see the difference between peak and min as jitter, and smooth to peak delay.

Two TCP Flows (Effects of Phase)



Green is flow 1, Blue is flow 2. Both do identical AIMD.

Left: sawtoothes in phase. Right: same sawtoothes, out of phase.

Multiple TCP flows and Queues

- If multiple flows all back-off in phase, the router still needs a delay-bandwidth product of buffering.
- If multiple flows back-off out of phase, high utilization can be maintained with smaller queues.
  - How to keep the flows out of phase?

Active Queue Management

Goals of Active Queue Management

- The primary goal: Controlling average queuing delay, while still maintaining high link utilization.
- Secondary goals:
  - Improving fairness (e.g., by reducing biases against bursty low-bandwidth flows).
  - Reducing unnecessary packet drops.
  - Reducing global synchronization (i.e., for environments with small-scale statistical multiplexing).
  - Accommodating transient congestion (lasting less than a round-trip time).

Random Early Detection (RED)

- As queue builds up, randomly drop or mark packets with increasing probability (before queue gets full).
- Advantages:
  - Lower average queuing delay.
  - Avoids penalizing streams with large bursts.
  - Desynchronizes co-existing flows.

Original RED Algorithm

```
for each packet arrival
    calculate the new average queue size q_avg
    if min_th < q_avg < max_th
        calculate probability p_a
        with probability p_a:
            mark/drop the arriving packet
    else if max_th < q_avg
        drop the arriving packet
```

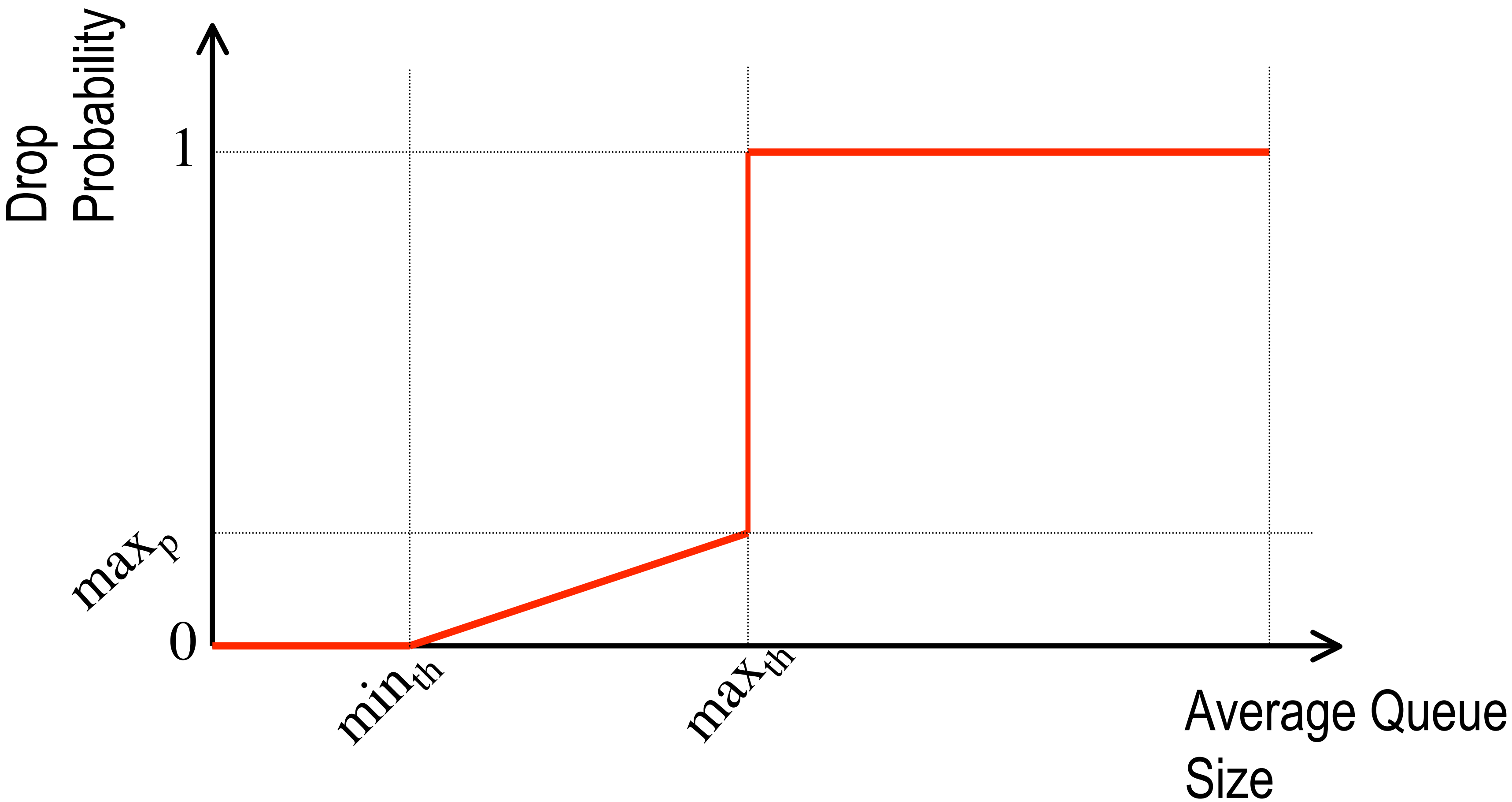
Variables:

- q_{avg} : average queue size
- p_a : packet marking or dropping probability

Parameters:

- min_{th} : minimum threshold for queue
- max_{th} : maximum threshold for queue

RED Drop Probabilities



The argument for using the *average* queue size in AQM

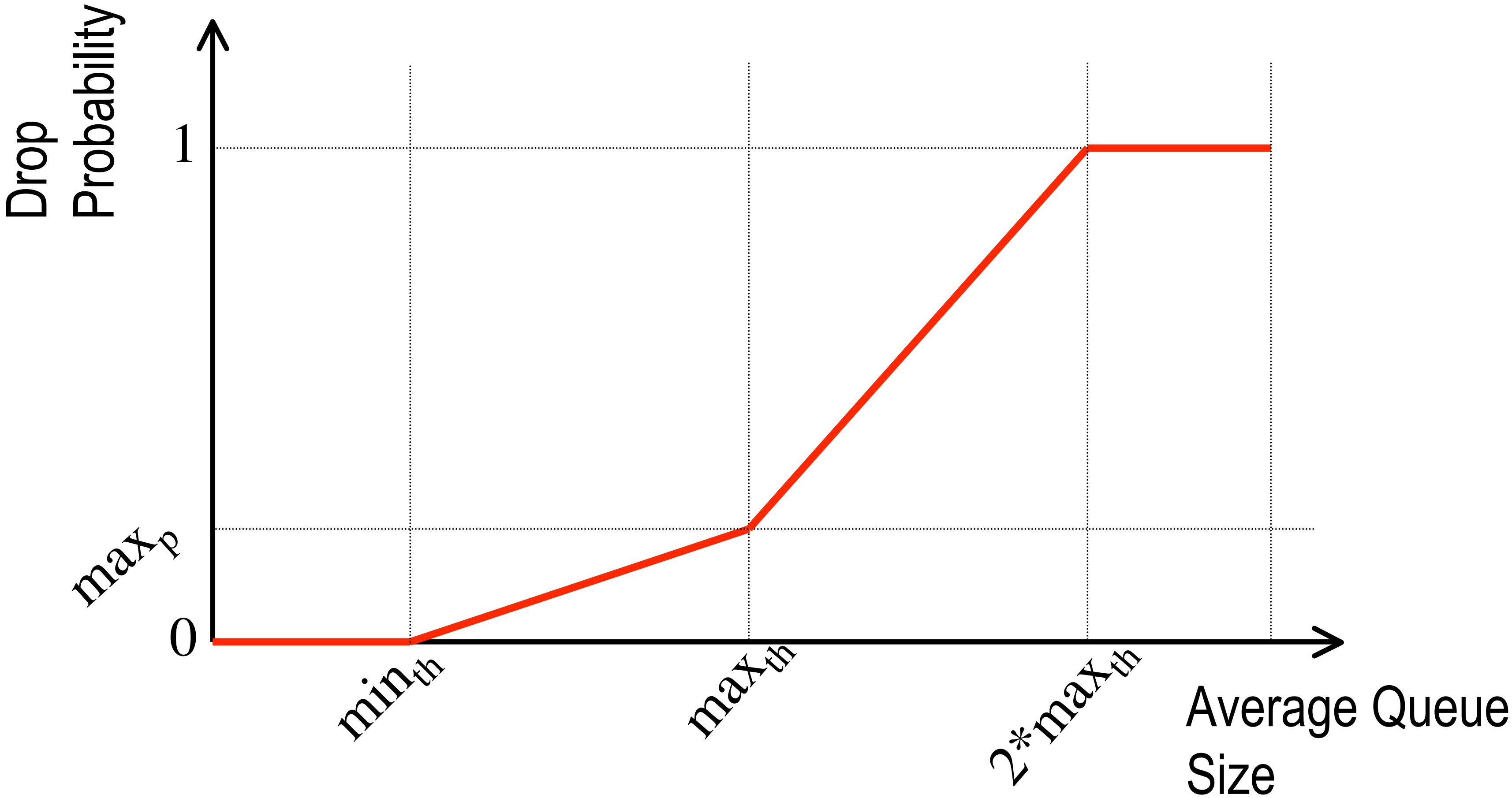
To be robust against transient bursts:

- When there is a transient burst, to drop just enough packets for end-to-end congestion control to come into play.
- To avoid biases against bursty low-bandwidth flows.
- To avoid unnecessary packet drops from the transient burst of a TCP connection slow-starting.

The problem with RED

- Parameter sensitivity
  - How to set min_{th}, max_{th} and max_p?
- Goal is to maintain mean queue size below the midpoint between min_{th} and max_{th} in times of normal congestion.
  - max_{th} needs to be significantly below the maximum queue size, because short-term transients peak well above the average.
  - max_p primarily determines the drop rate. Needs to be significantly higher than the drop rate rfequired to keep the flows under control.
- In reality it's hard to set the parameters robustly, even if you know what you're doing.

RED Drop Probabilities (Gentle Mode)



Other AQM schemes.

- Adaptive RED (ARED)
- Proportional Integral (PI)
- Virtual Queue (VQ)
- Random Exponential Marking (REM)
- Dynamic-RED (DRED)
- Blue
- Many other variants... (a lot of PhDs in this area!)

AQM: Summary

Multimedia traffic has tight delay constraints.

- Droptail queuing gives unnecessarily large queuing delays if good utiilization is needed.
- Packet loss as a signal of congestion hurts real-time traffic much more than it hurts file transfer.
  - No time to retransmit.

AQM combined with ECN can give low loss, low-ish delay, moderate jitter service.

- No admission control or charging needed.
- But no guarantees either - it's still best-effort.

Part 2

Congestion control for real-time traffic.

New Applications

TCP continues to serve us well as the basis of most transport protocols, but some important applications are not well suited to TCP:

- Telephony and Video-telephony.
- Streaming Media.
- Multicast Applications.

TCP is a reliable protocol. To achieve reliability while performing congestion control means trading delay for reliability.

- Telephony and streaming media have limited delay budgets - they don't want total reliability.
- TCP cannot be used for multicast because of response implosion issues (amongst other problems).

Non-TCP Congestion Control.

We can separate TCP's congestion control (AIMD) from TCP's reliability mechanism.

- Eg: RAP (Rate Adaptation Protocol) Rejaie et al, Infocom 1999.

However, AIMD congestion control gives a flow throughput that *changes very rapidly*, which is not well suited to streaming applications that want to delivery consistent quality to the end-user.

- Streaming playback from servers can work around this using *receiver buffering* (Eg: Rejaie et al, Sigcomm 1999), but it would be better to have a congestion control scheme that was less variable in the first place.

TCP-Friendly

- Any alternative congestion control scheme needs to *coexist with TCP in FIFO queues* in the best-effort Internet, or *be protected from TCP* in some manner.

- To co-exist with TCP, it must impose the same long-term load on the network:
  - No greater long-term throughput as a function of packet loss and delay so TCP doesn't suffer
  - Not significantly less long-term throughput or it's not too useful

Solution Space: Unicast Streaming Media

1. AIMD with different constants
   - Eg: increase window by $^3/_7$ packets each RTT, decrease multiplicatively by $^3/_4$ when a loss occurs.

2. Equation-based congestion control.
   - Try to design a protocol to achieve the throughput as TCP does on medium timescales.

TFRC: General Idea

Use a model of TCP's throughout as a function of the loss rate and RTT directly in a congestion control algorithm.

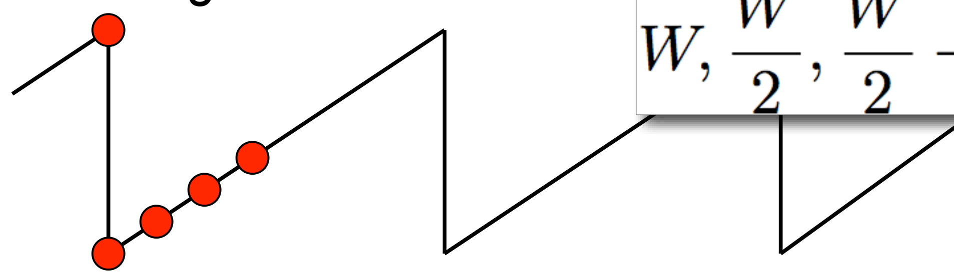
- If transmission rate is higher than that given by the model, reduce the transmission rate to the model's rate.
- Otherwise increase the transmission rate.

TCP Modelling: The "Steady State" Model

The model: Packet size B bytes, round-trip time R secs, no queue.

- A packet is dropped each time the window reaches W packets.
- TCP's congestion window:

$$W, \frac{W}{2}, \frac{W}{2}+1, \frac{W}{2}+2, \ldots, W, \frac{W}{2}, \ldots$$



- The maximum sending rate in packets per roundtrip time: W
- The maximum sending rate in bytes/sec: $W\,B\,/\,R$
- The average sending rate T: $T = (3/4)W\,B\,/\,R$
- The packet drop rate p: $p = \frac{1}{\frac{3}{8}W^2}$
- **The result:** $T = \frac{\sqrt{6}B}{2R\sqrt{p}} = \frac{\sqrt{3/2}B}{R\sqrt{p}}$

An Improved "Steady State" Model

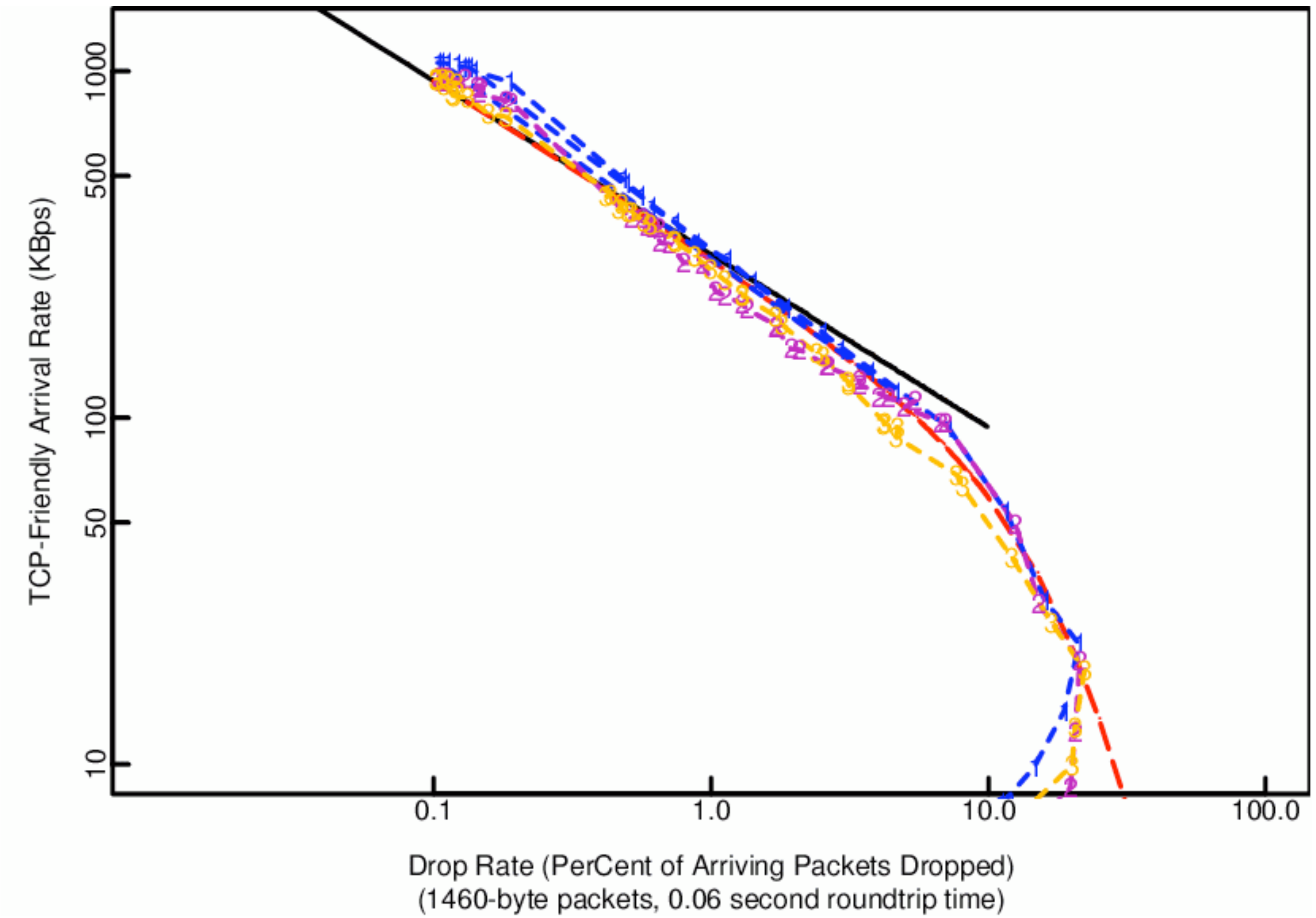
A pretty good improved model of TCP Reno, including timeouts, from Padhye et al, Sigcomm 1998:

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}$$

T : sending rate in bytes/second]
R : round trip time
p : fraction of packets lost
t_{RTO} : TCP retransmission timeout

Would be better to have a model of TCP SACK, but the differences aren't critical.

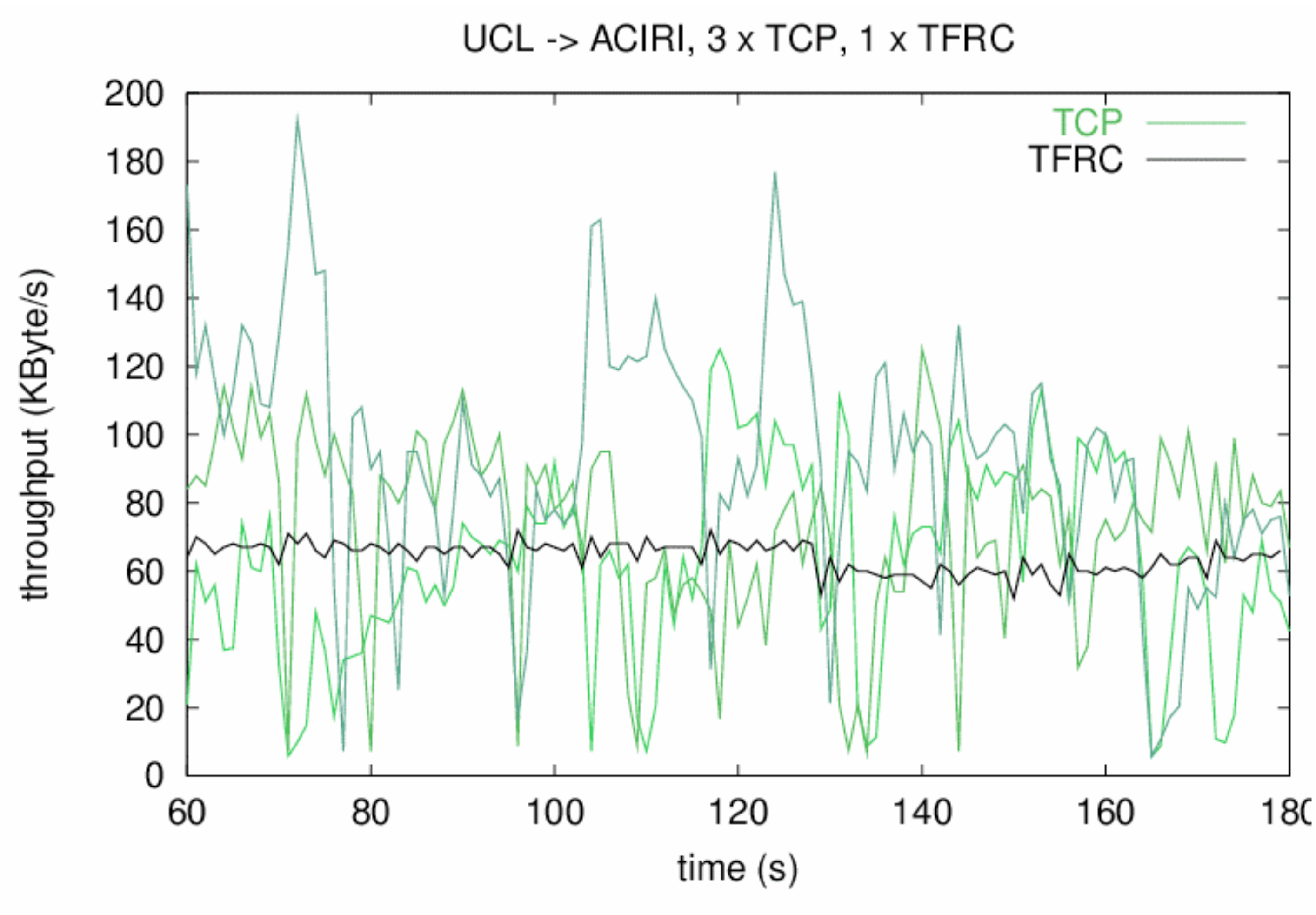
Verifying the Models



TFRC Details

- The devil's in the details.
  - How to measure the loss rate?
  - How to use RTT and prevent oscillatory behavior?
- Not as simple as we first thought.
- For the details, see:
  - Sally Floyd, Mark Handley, Jitendra Padhye, and Joerg Widmer, *Equation-Based Congestion Control for Unicast Applications,* Proc ACM SIGCOMM 2000.

TFRC Performance (Experimental)



Datagram Congestion Control Protocol (DCCP)

- Implementing congestion control correctly is hard.
- It's not usually the area of expertise of the application writer, and certainly doesn't get their product to market faster.
- TCP is a non-starter.
- UDP has problems getting though firewalls and NATs because it's connectionless.

"How about providing a protocol to help out the application writers, and give them some incentive to do the right thing?"

- Result: DCCP.

DCCP

The *Datagram Congestion Control Protocol* (DCCP) is a new minimalist ``transport'' protocol for apps that care more about delay than reliability.

- Allows negotiation of different congestion control algorithms.
- Provides a simple base on top of which more complex protocols can be built.
- Explicit connection setup/teardown helps NATs and firewalls.

DCCP Congestion Control

DCCP supports negotiation of the congestion control mechanism. Two CCIDs currently specified:

CCID 2: TCP-like congestion control.

- AIMD without TCP's reliability
- For applications that can tolerate AIMD's sawtooth behaviour and rapid changes in bandwidth.
- Advantages: rapid reaction lowers loss rate, quickly takes advantage of available capacity.

CCID 3: TFRC congestion control.

- For applications where smoothness and predictability is most important.

DCCP status

- RFC 4340, March 2006
- Currently a few implementations, shipping in Linux.
- Operating system APIs still a work-in-progress.
- Not clear yet if it will ever become commonplace enough for application writers, firewalls and NATs to assume it’s existence.

Applications and Congestion Control

- *"What's in it for me?"*
  - Why would an multimedia application writer choose to add congestion control?
  - Disadvantages:
    - Extra Complexity.
    - Get to go slower.
    - Variable quality may annoy users.
- "*I can just add all this redundancy and FEC you told me about to protect my flows from packet loss.*"
- *"If I don't adapt my rate, all those adaptive TCP flows will just be nice and get out of my way!"*

Remaining Problems

Remaining Problems

- TFRC (or something similar) for applications that need a constant rate in packets per second.
- TFRC for applications that can only send at certain fixed rates.
- Congestion control for lossy links.
  - Loss does not always imply congestion.
- Insufficient dynamic range
  - Wide-area, high speed.
- Quick startup.
- Low delay
- Overall concept of fairness (eg. BitTorrent)

Remaining Problems

- TFRC for applications that need a constant rate in packets per second.
- TFRC for applications that can only send at certain fixed rates.
- Congestion control for lossy links.
  - Loss does not always imply congestion.
- Insufficient dynamic range
  - Wide-area, high speed.
- Quick startup.
- Low delay
- Overall concept of fairness (eg. BitTorrent)

Part 3

High-speed congestion control

AIMD: Insufficient Dynamic Range

- In steady state, TCP throughput is approximately:

$$Throughput = \frac{packetsize}{RTT\sqrt{p_{loss}}}$$

- Transmitting at high rate across high-latency links requires:
  - a very large congestion window
  - a very low loss rate
  - a very long time to converge to fairness.

High Delay-Bandwidth Products

Assume one loss every half hour, 200ms RTT, 1500bytes/pkt.

How fast can we go?

⇒ 9000 RTTs increase between losses.

⇒ peak window size = 18000 pkts.

⇒ mean window size = 12000 pkts.

⇒ 18MByte/RTT

⇒ 720Mbit/s.

⇒ Needs a bit-error rate of better than 1 in 10^12.

⇒ Takes a very long time to converge or recover from a burst of loss.

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Tan, MSFT)

Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP etc.

Based on modified routers

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Tan, MSFT)

Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP etc.

Based on modified routers

High-speed TCP

Additive-increase, multiplicative-decrease:

- No Loss, each RTT:
  - $w = w + a$
- Loss, each RTT:
  - $w = w - bw$

For regular TCP, $a = 1$, $b = 0.5$.

General idea for High-speed TCP: as w increases, increase a and decrease b to make TCP less sensitive to loss.

- Do this to change the response function so that $W = 0.12/p^{0.835}$
- At low speeds, do the same as regular TCP so that it's backwards compatible.

Changing a and b

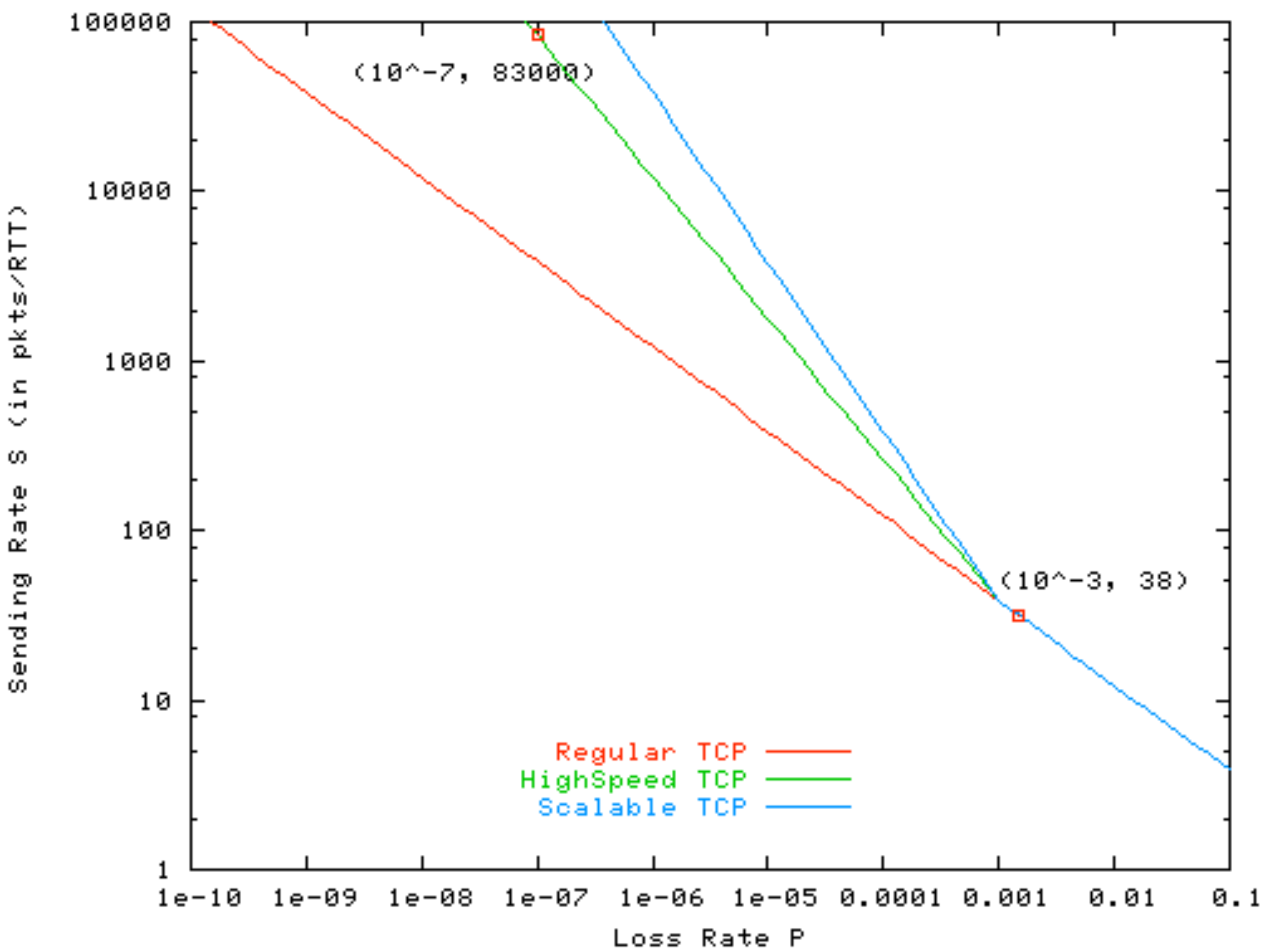
As the window increases:

- *a* is increased progressively
- a corresponding value of *b* is calculated so as to track the desired response curve.

w	a(w)	b(w)
38	1	0.50
118	2	0.44
221	3	0.41
347	4	0.38
495	5	0.37
663	6	0.35
851	7	0.34
1058	8	0.33
1284	9	0.32
1529	10	0.31
1793	11	0.30
2076	12	0.29
2378	13	0.28

[Source: Sally Floyd]

High-speed TCP (Floyd)



[Source: Sally Floyd]

How does this work in reality?

Bandwidth	Avg Cwnd w (pkts)	Increase *a* (pkts)	Decrease *b* (w)
1.5 Mbps	12.5	1	0.50
10 Mbps	83	1	0.50
100 Mbps	833	6	0.35
1 Gbps	8333	26	0.22
10 Gbps	83333	70	0.10

Values are for a network with 100ms RTT, 1500 byte packets.

[Source: Sally Floyd]

High-speed TCP

Advantages:

- Simple changes to TCP
- Backwards compatible with existing TCP
- Requires no infrastructure change.

Disadvantages:

- Same needs as TCP for large amounts of buffering in queues
  - Not good for low-delay multimedia, games, etc.
- Not infinitely scalable.

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Tan, MSFT)

Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP etc.

Based on modified routers

Scalable TCP (Kelly)

Similar to high-speed TCP:

- Uses a fixed decrease parameter *b* of 1/8
- Uses a fixed increase per acknowledgement of 0.01.
  - Gives an increase parameter *a* of 0.005 *w* per window.

- The effect is a constant number of RTTs between lost packets.
  - Thus scale-invariant.

Scalable TCP

Advantages:

- As with High-speed TCP
- Scalable to any link speeds

Disadvantages:

- Same needs as TCP for large amounts of buffering in queues.
- Possible issues with convergence if drop-tail queues cause flows to synchronize their backoffs

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Microsoft)

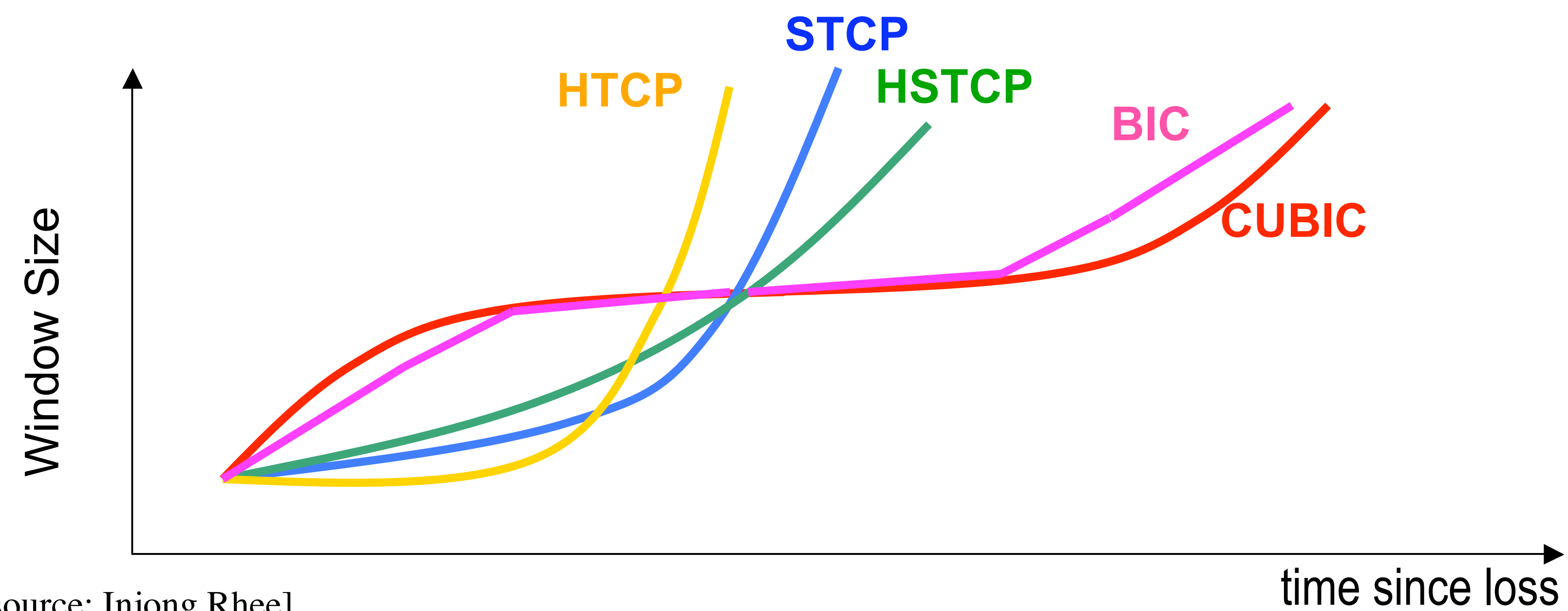
Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP etc.

Based on modified routers

Cubic

- Different high-speed TCP variants propose different response functions.
  - Function of time since last loss?
  - Function of current window size?

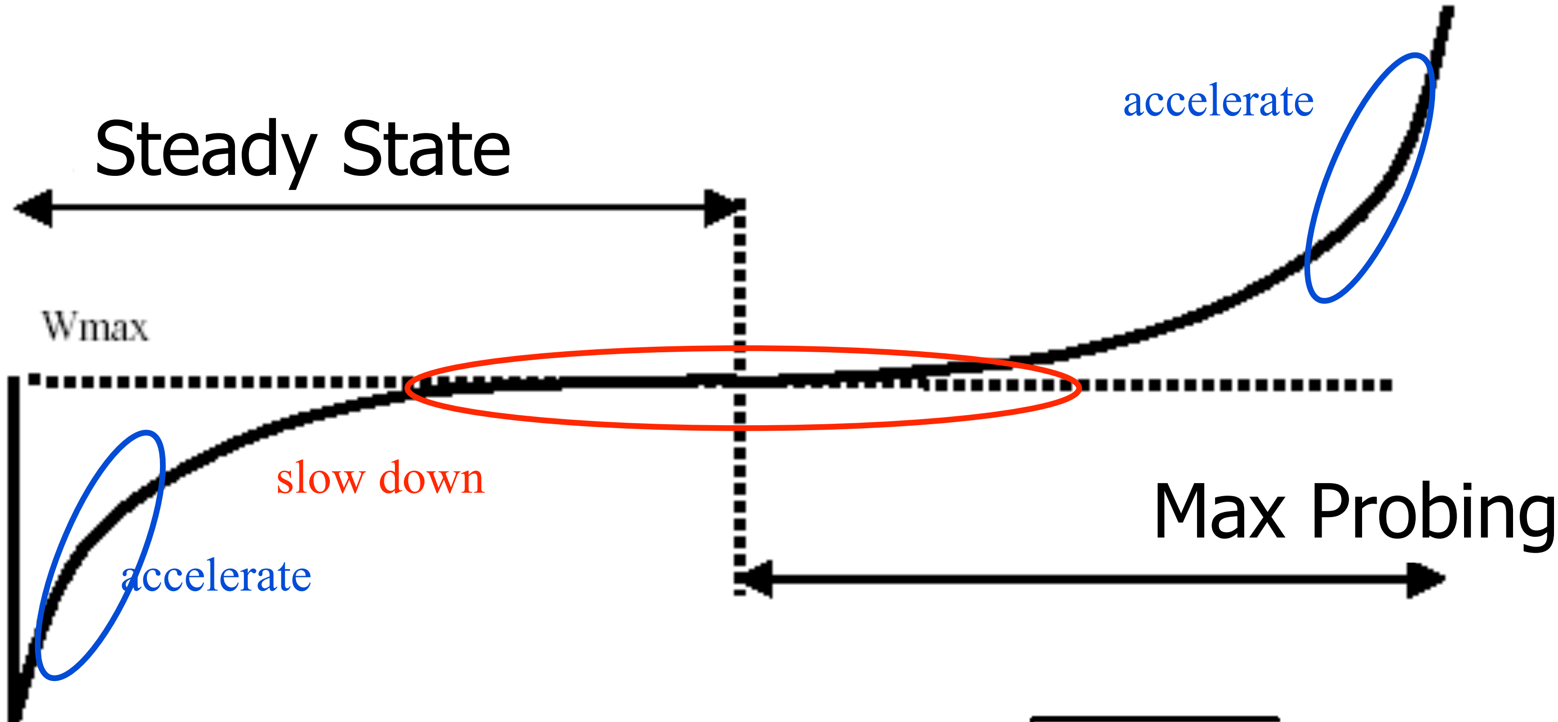


[source: Injong Rhee]

Cubic: basic idea

- Keep track of maximum window recently used (W_{max}).
  - Increase quickly immediately after a loss.
  - Increase slowly as W_{max} approaches.
  - Increase steadily more quickly as W_{max} is left behind.
- Motivation:
  - If network conditions are unchanged, want to spend a long time around W_{max}.
  - If conditions have changed, want to find new operating point quickly.

CUBIC Window Growth Function



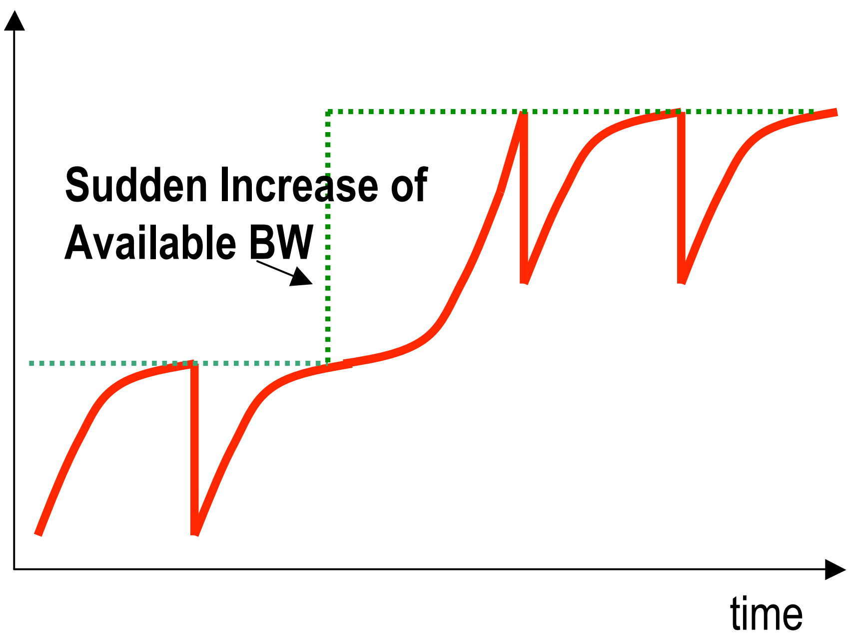
$$W_{cubic} = C(t-K)^3 + W_{max} \qquad K = \sqrt[3]{W_{max}\beta / C}$$

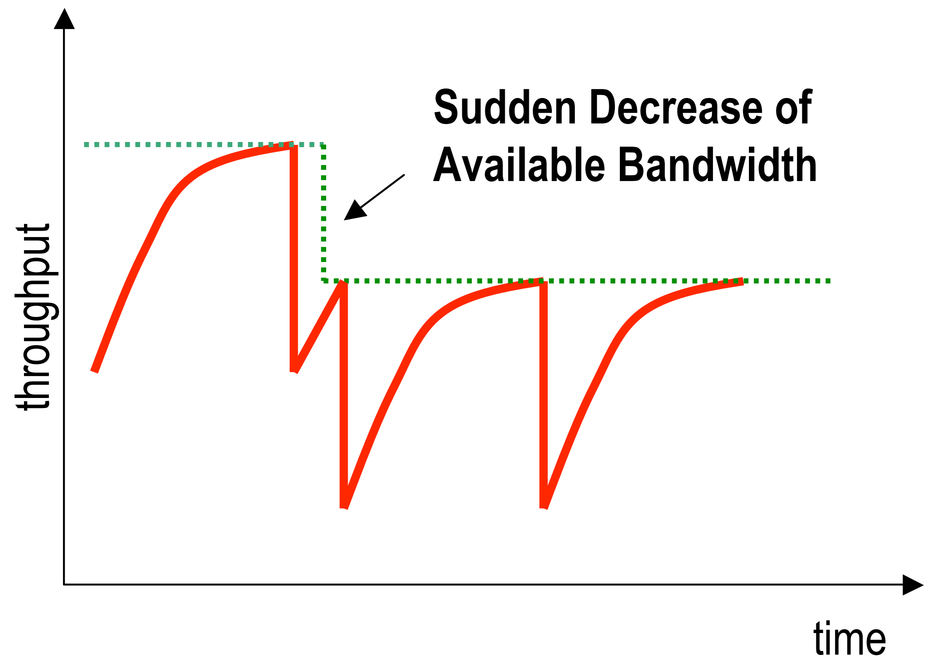
where ***C*** is a scaling factor, ***t*** is the elapsed time from the last window reduction, and ***β*** is a constant multiplication decrease factor

[source: Injong Rhee]

Concave/convex functions

- History information from previous epoch will often be good.
- But adapt when it is wrong.





[source: Injong Rhee]

Cubic in Linux

- Cubic is the default congestion control algorithm in Linux.
  - Not clear how this decision was made.
  - Not clear how nicely Cubic plays with other high-speed variants.

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Tan, MSFT)

Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP etc.

Based on modified routers

FAST (S. Low et al.)

FAST uses **delay** as the principle way to sense congestion.

Advantages:

- Delay gives multi-bit feedback per RTT.
- Delay is an *early* congestion signal.

Disadvantages:

- Hard to co-exist with existing TCP, or DoS attacks.
- Congestion signal can be noisy, or confused by variable latency links such as 802.11
- Delay as a congestion signal tends to saturate when there are many flows sharing a bottleneck.

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Tan, MSFT)

Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP etc.

Based on modified routers

Compound TCP

Motivation: FAST is good at increasing the rate rapidly when the net is underutilized, but doesn't play well with vanilla TCP when the net is congested.

Can we get the best of both worlds?

- Regular AIMD behaviour driven by losses.
- When the net is underutilized, use a delay-based metric to increase very rapidly.
- As the net becomes congested, move progressively from one regime to the other.

CTCP

Actual window $win = cwnd + dwnd$

- Adapt *cwnd* pretty much as with regular TCP: AIMD.
- Adapt *dwnd* each RTT:

Estimate Q, the number of packets backlogged

if $Q < thresh$

$dwnd \mathrel{+}= \max(\alpha \cdot win^k - 1,\ 0)$

else

$\mathrm{dwnd} \mathrel{-}= \zeta \cdot Q$

rapid increase when unloaded

reduce *dwnd* as *cwnd* builds queue

On loss: $dwnd = \max(win\,(1\text{-}\beta)\text{-}cwnd/2,\ 0)$

CTCP

- **Advantages:**
  - Rapid increase to use spare capacity.
  - Plays fair with regular TCP
  - Degradation to regular TCP when delay metric gets confused (eg wireless, etc)
- **Disadvantages:**
  - Bursty reverse-path traffic can incorrectly push Q estimate over threshold, slowing throughput.
  - Convergence to fairness is primarily from $cwnd$ AIMD, so can be slow.

Windows Vista

- Compound TCP ships in Windows Vista.
  - Not enabled by default, but there for anyone who needs to operate over very high delay-bandwidth product networks.

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Microsoft)

Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP etc.

Based on modified routers

Explicit Congestion Control

Thought experiment:

- What if you put the current window and RTT in every packet, so the routers know what was going on?
- What if you allowed the routers to signal exactly how a flow should change its window?

But still keep the routers stateless (no per-flow state).

XCP

From packet headers, routers can estimate the mean RTT, mean window, and number of flows.

Routers can calculate the per-packet delta in window needed to converge to optimal utilization.

Routers can divide up the total delta so that different flows converge to the same throughput, regardless of RTT.

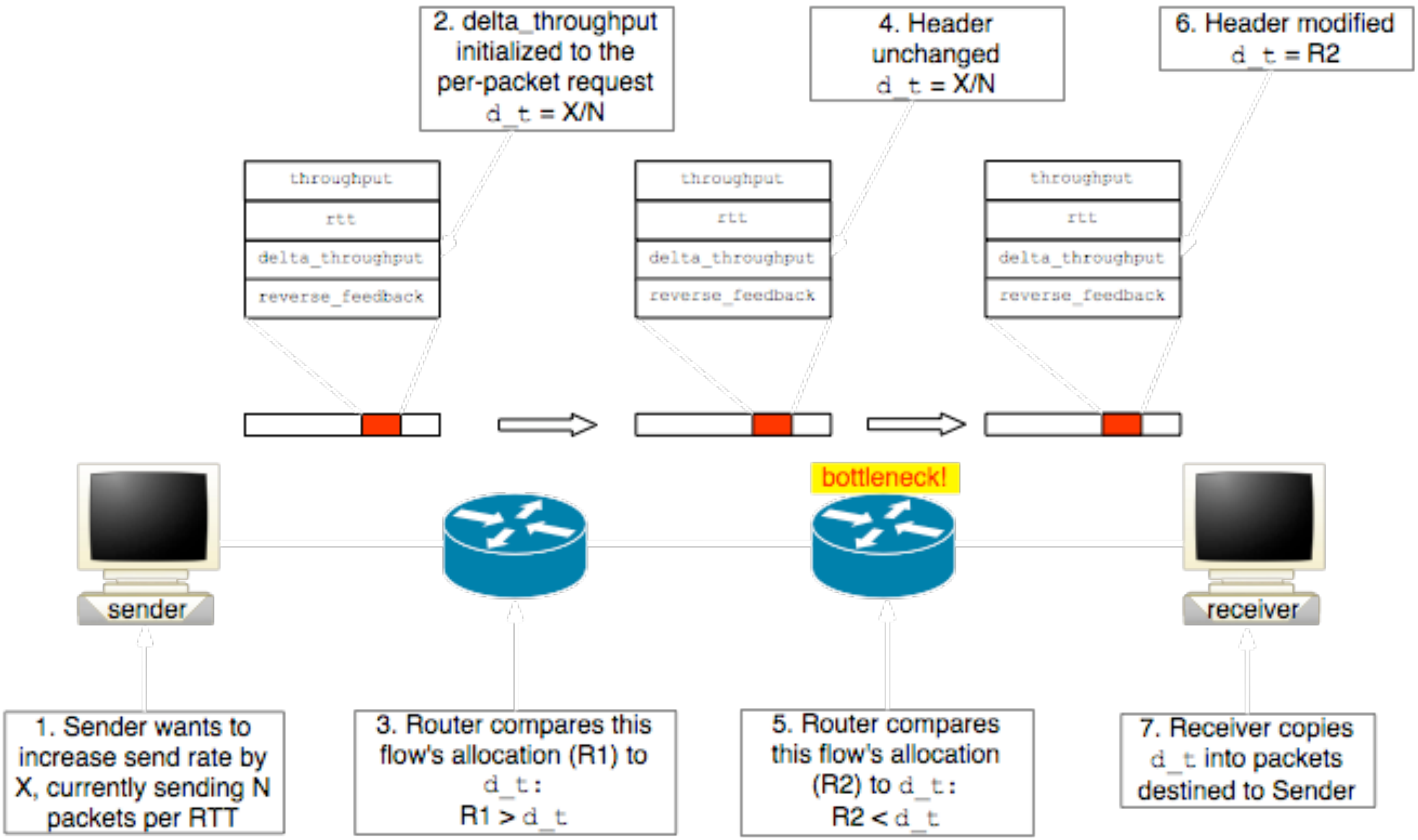
"Internet Congestion Control for High Bandwidth-Delay Product Environments", D.Katabi, M.Handley & C.Rohrs, Sigcomm 2002.

Explicit signaling happens via the congestion header

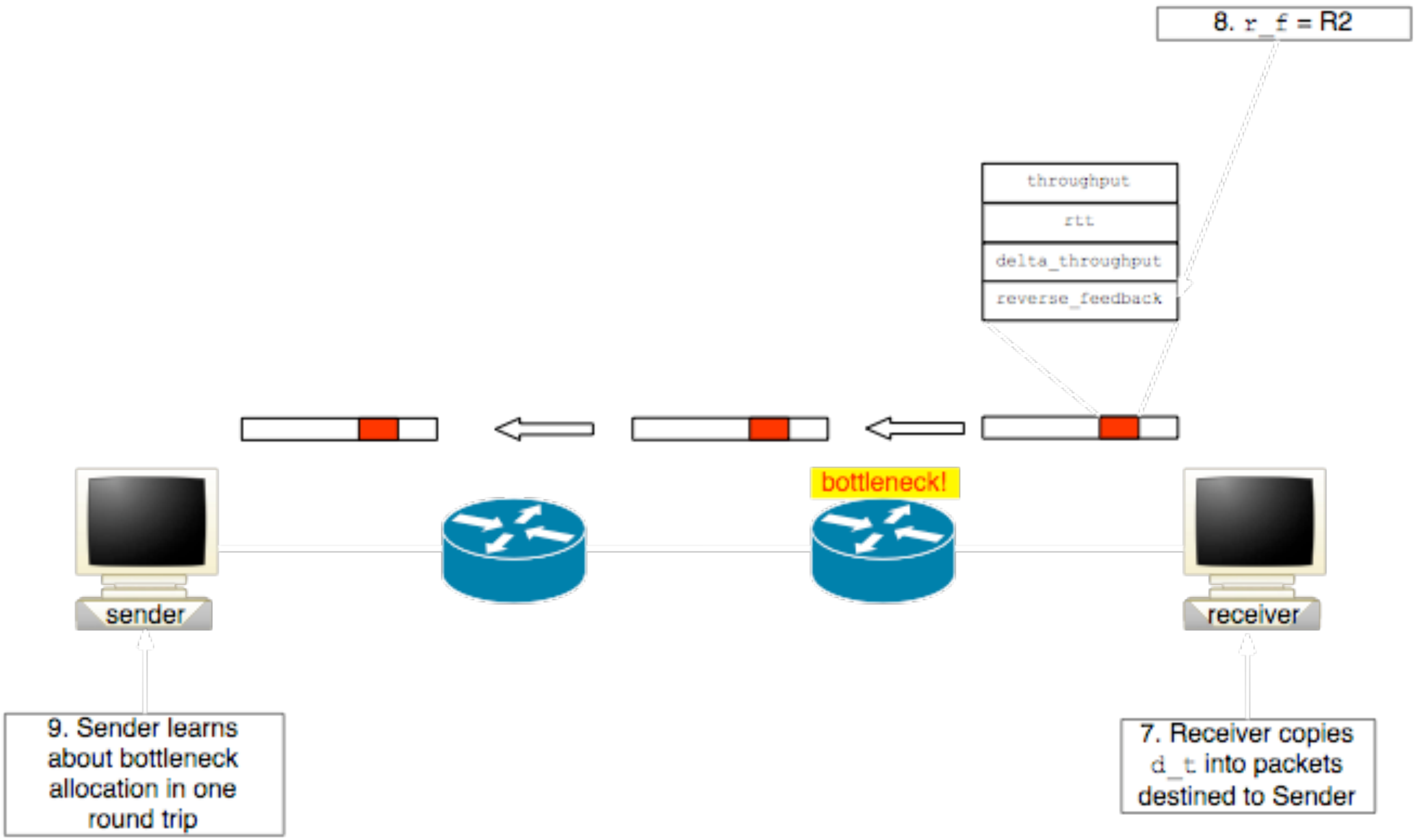
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|version|format |   protocol    |    length     |    unused     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              rtt                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          throughput                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       delta_throughput                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       reverse_feedback                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

[figure from Aaron Falk at ISI]

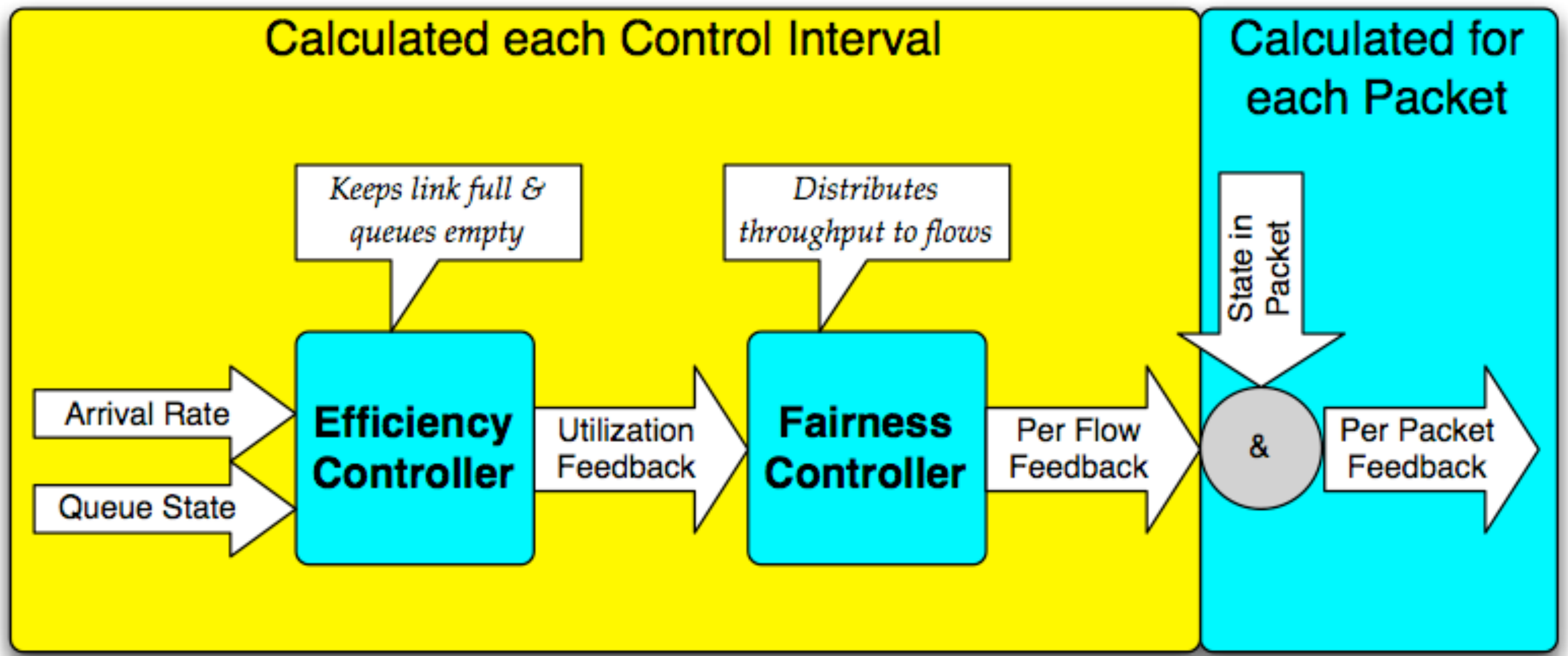
XCP Feedback Loop



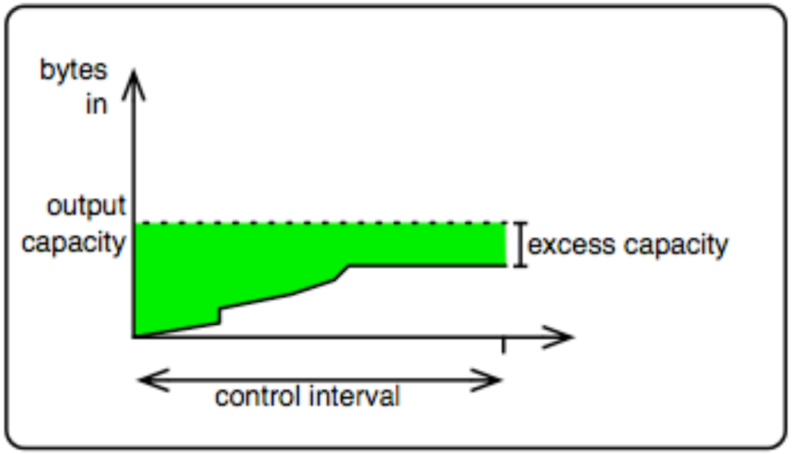
XCP Feedback Loop

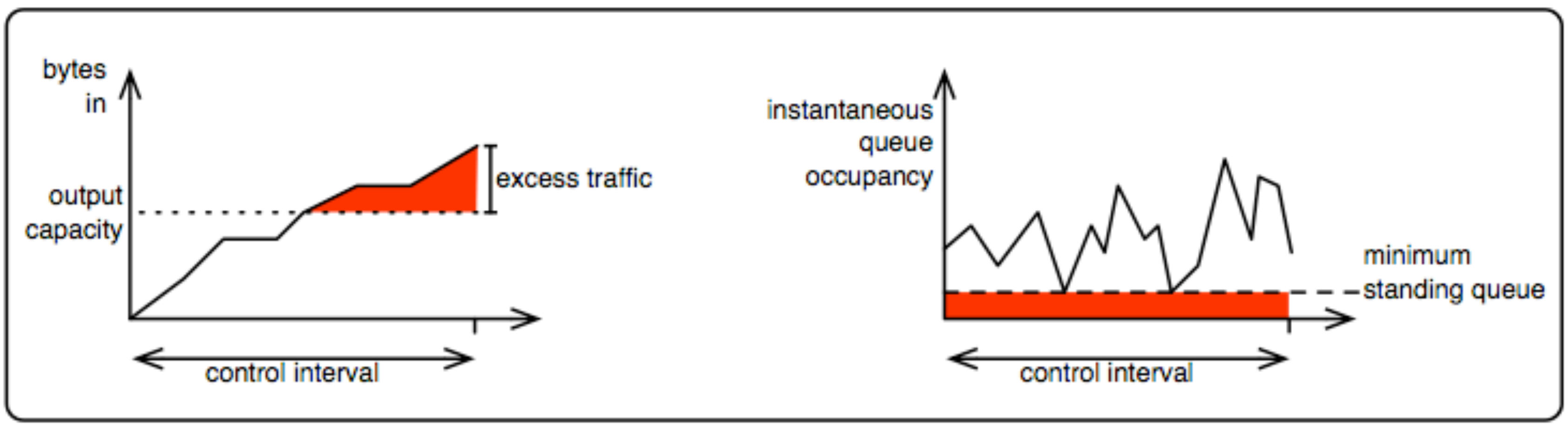


Efficiency and Fairness Algorithms are Independent



Utilization Feedback is derived from Arrivals and Queue

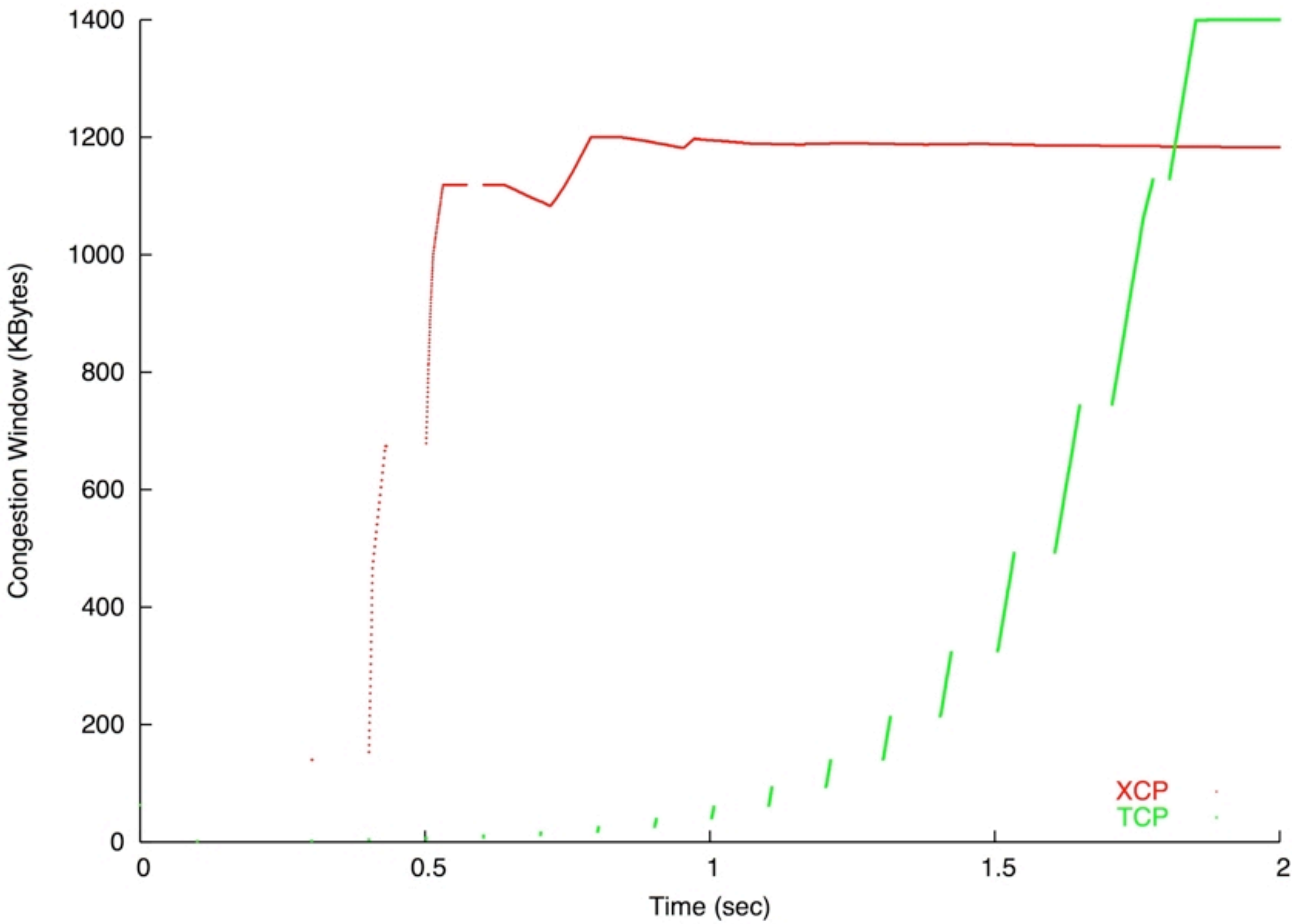




Benefits of XCP

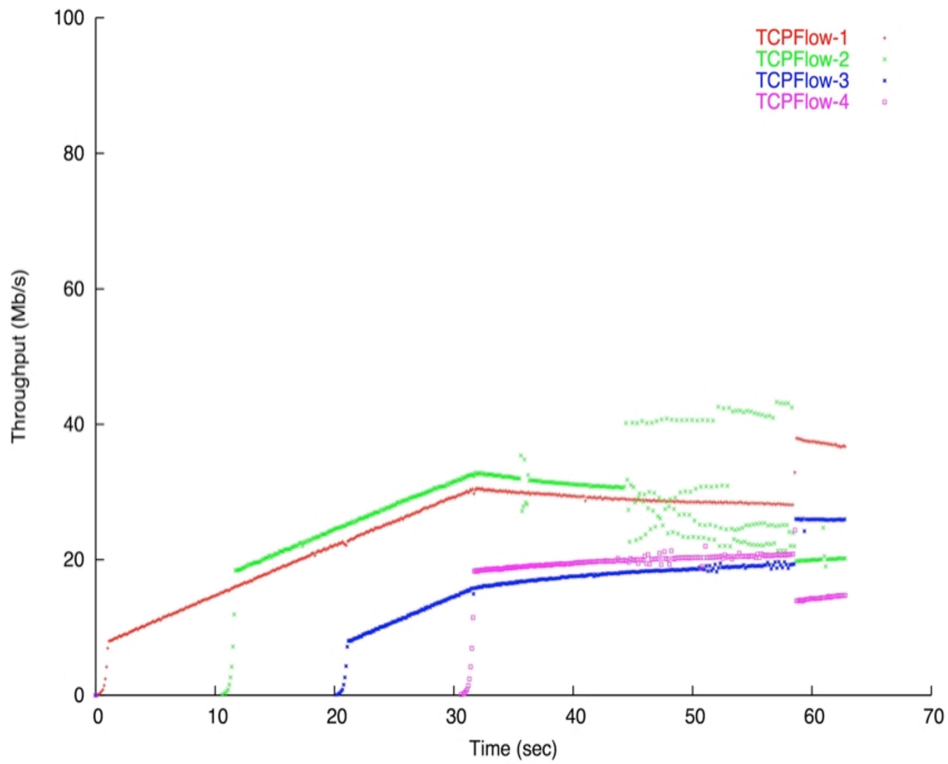
- In simulation...
  - XCP fills the bottleneck pipe much more rapidly than AIMD congestion control.
  - XCP rapidly converges to fair allocation of bottleneck bandwidth.
  - XCP gets better bottleneck link utilization than VJCC for large bandwidth-delay-product flows.
  - XCP maintains tiny queues
  - XCP is more stable than VJCC at long RTTs

XCP vs. TCP startup behavior

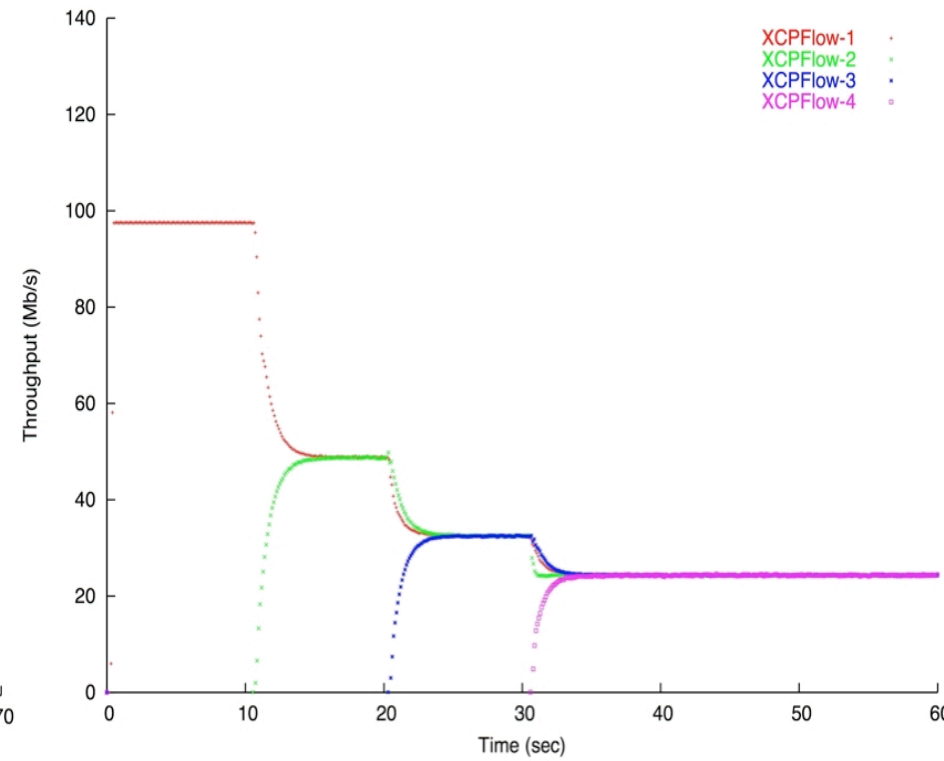


[results from Aaron Falk at ISI]

Comparing TCP & XCP Throughput

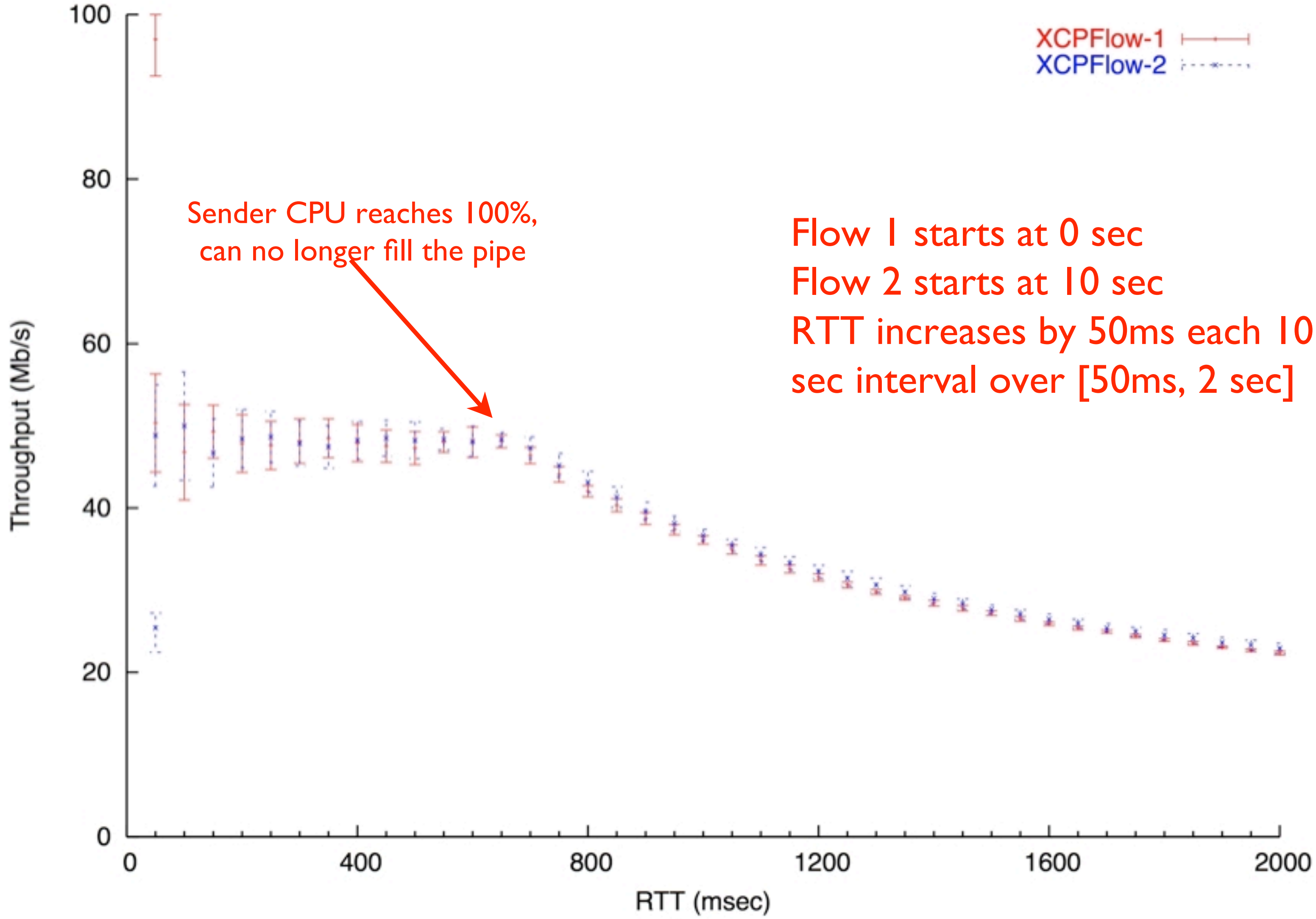


TCP Measured



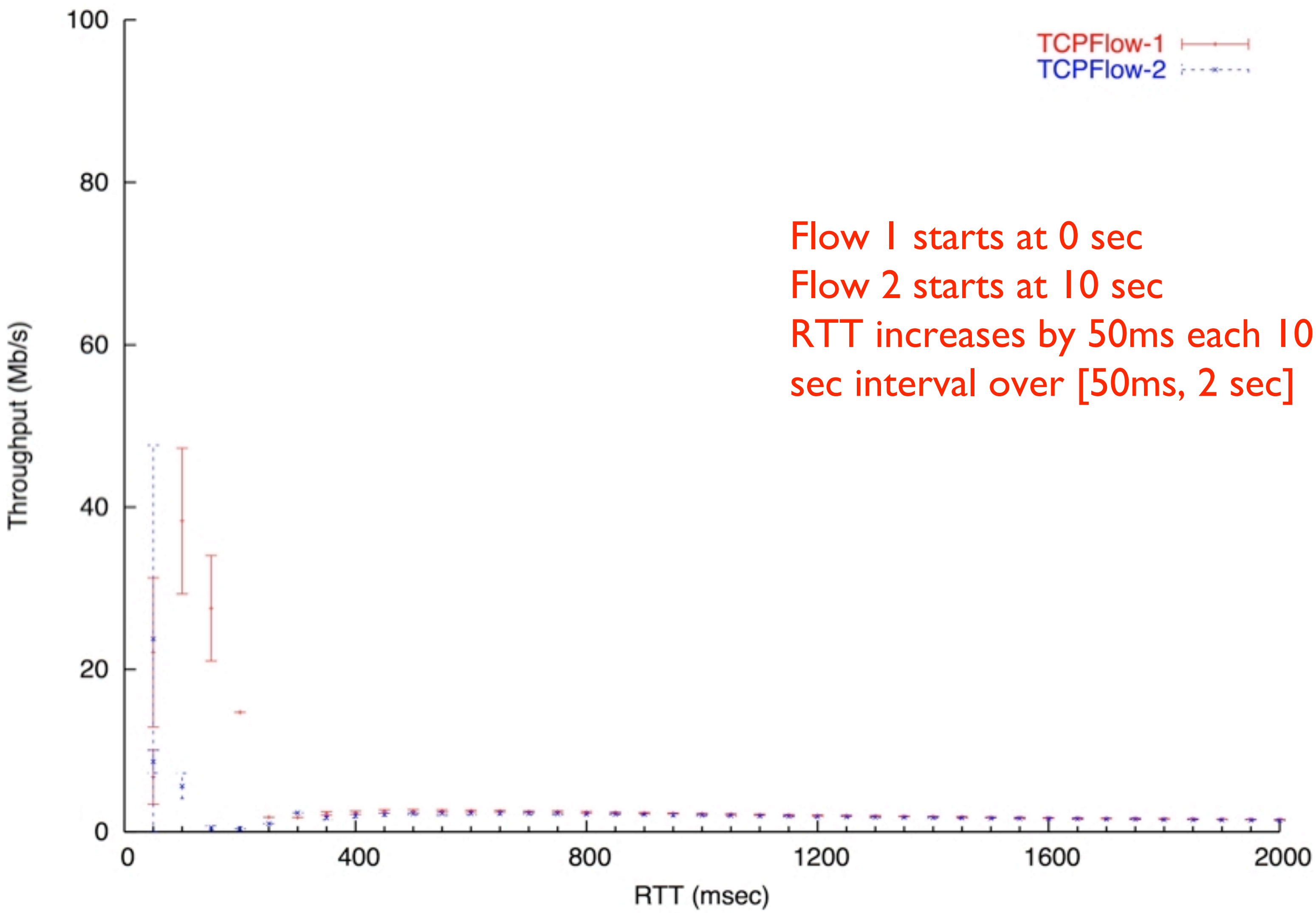
XCP Measured

XCP is stable as the RTT increases



Flow 1 starts at 0 sec
Flow 2 starts at 10 sec
RTT increases by 50ms each 10 sec interval over [50ms, 2 sec]

TCP doesn't do as well…



Flow 1 starts at 0 sec
Flow 2 starts at 10 sec
RTT increases by 50ms each 10 sec interval over [50ms, 2 sec]

XCP Experiments Summary

- Early measurements match simulated results
- XCP fairly allocates bottleneck bandwidth to multiple flows
- XCP dynamically reallocates bottleneck bandwidth as flows arrive and depart
- XCP remains stable as RTT varies by 4000%

XCP

Advantages:

- Rapid convergence.
- Low loss, low delay.
- Can compensate for RTT differences.

Disadvantages:

- Many bits needed in each packet.
- Moderately expensive in routers.
- Needs all routers/switches to be modified.

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Microsoft)

Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP etc.

Based on modified routers

VCP

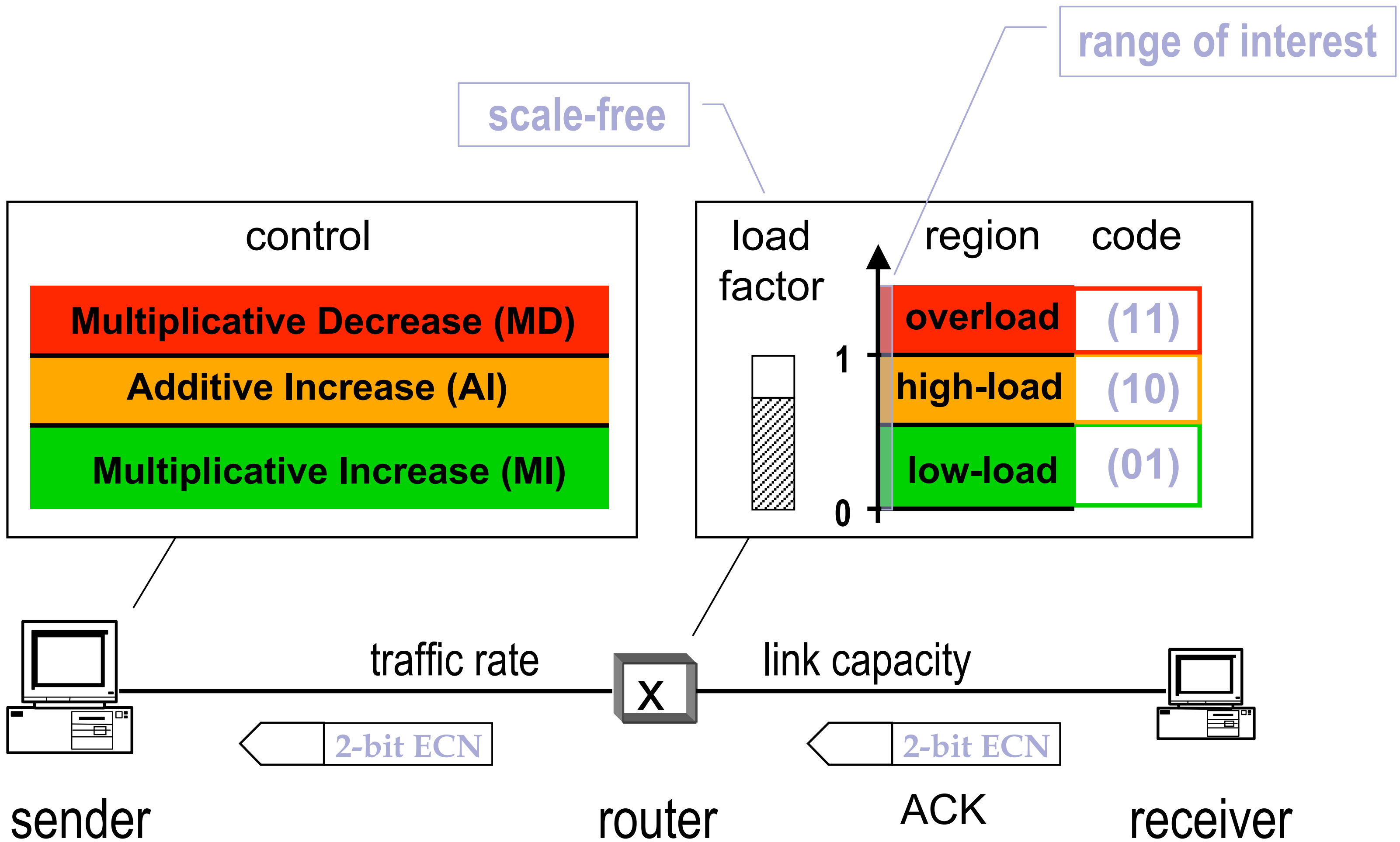
"Variable-structure congestion Control Protocol"

- XCP uses separates efficiency (utilization) control and fairness control. Both controlled by the routers.
- Observation:
  - *You don't care about fairness if the network is under-utilized.*

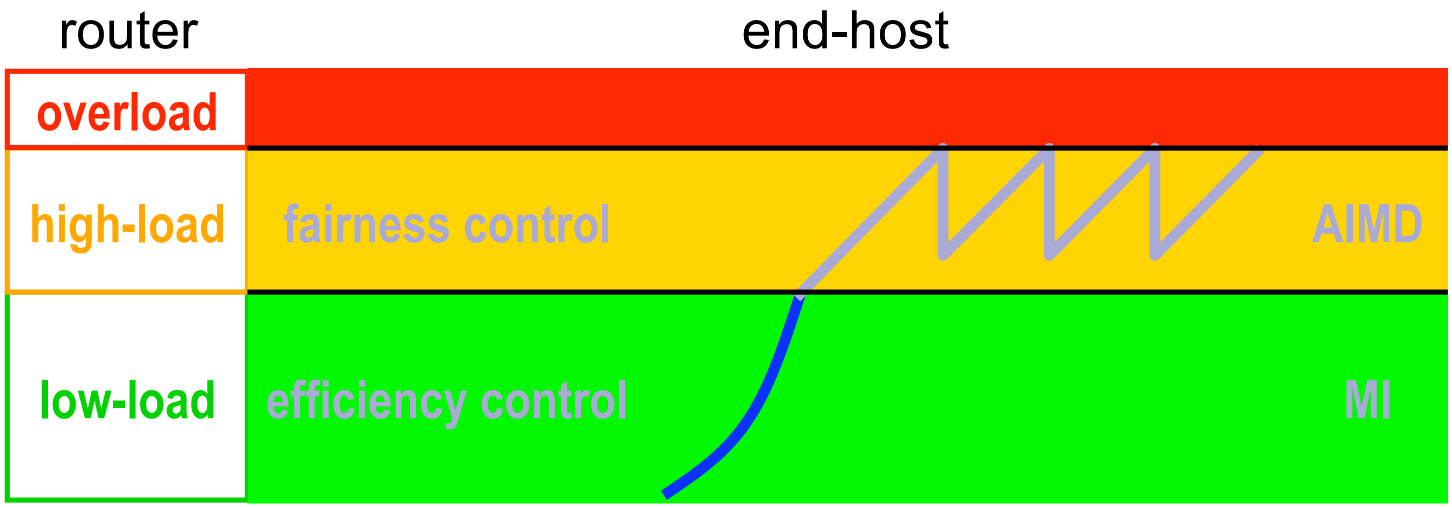
VCP:

- Routers signal *level of utilization*.
- End-systems *change modes*, from trying to maximize utilization to trying to maximize fairness as network reaches full utilization.

VCP


range of interest
scale-free
control
Multiplicative Decrease (MD)
Additive Increase (AI)
Multiplicative Increase (MI)
load
factor
region
code
overload
(11)
high-load
(10)
low-load
(01)
1
0
traffic rate
link capacity
X
2-bit ECN
2-bit ECN
sender
router
ACK
receiver

VCP



- Decouple efficiency and fairness controls in different load regions
- Use network link load factor as the congestion signal.
- Achieve high efficiency, low loss, and small queue
- Fairness model is similar to TCP:
  - Long flows get lower bandwidth than in XCP (proportional vs. max-min fairness)
  - Fairness convergence much slower than XCP

High-speed Congestion Control

- High-speed TCP (S. Floyd)
- Scalable TCP (T. Kelly)
- Cubic (I. Rhee)

Loss driven

- FAST (S. Low)
- Compound TCP (Microsoft)

Delay driven

- Fair queuing + packet pair (S. Keshav)
- ATM ABR service.
- XCP (D. Katabi)
- VCP (Y. Xia)
- RCP

Based on modified routers

Outline

Part 1: "Traditional" congestion control for bulk transfer.

Part 2: Congestion control for realtime traffic.

Part 3: High-speed congestion control.

Where next??

- Can we realistically change the routers to benefit congestion control?
- Big packets?
  - A congestion control scheme that increases the packet size (above a fixed number of packets in flight)?
- Per-flow queuing?
  - Provides improved isolation so many congestion control schemes can co-exist safely.

Summary

- This is a critical time for congestion control.
  - The mechanisms that have served us well for 15 years are starting to show their limitations.
  - The next few years will determine how we manage network resources for a long time.
  - There are many possible solutions, but all seem to have significant drawbacks.
  - There's no consensus on a solution right now, nor any process by which we might reach consensus.