

UiO : Faculty of Mathematics and Natural Sciences
University of Oslo

Department of Informatics
Networks and Distributed Systems (ND) group

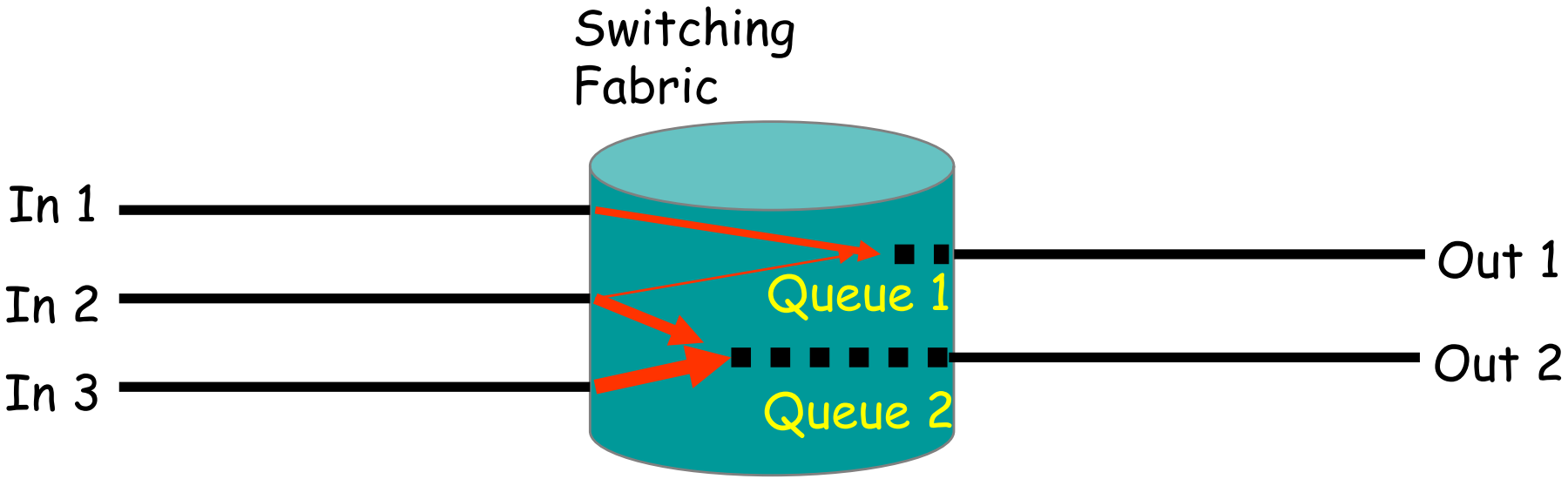
IN 3230 / IN 4230

Congestion control



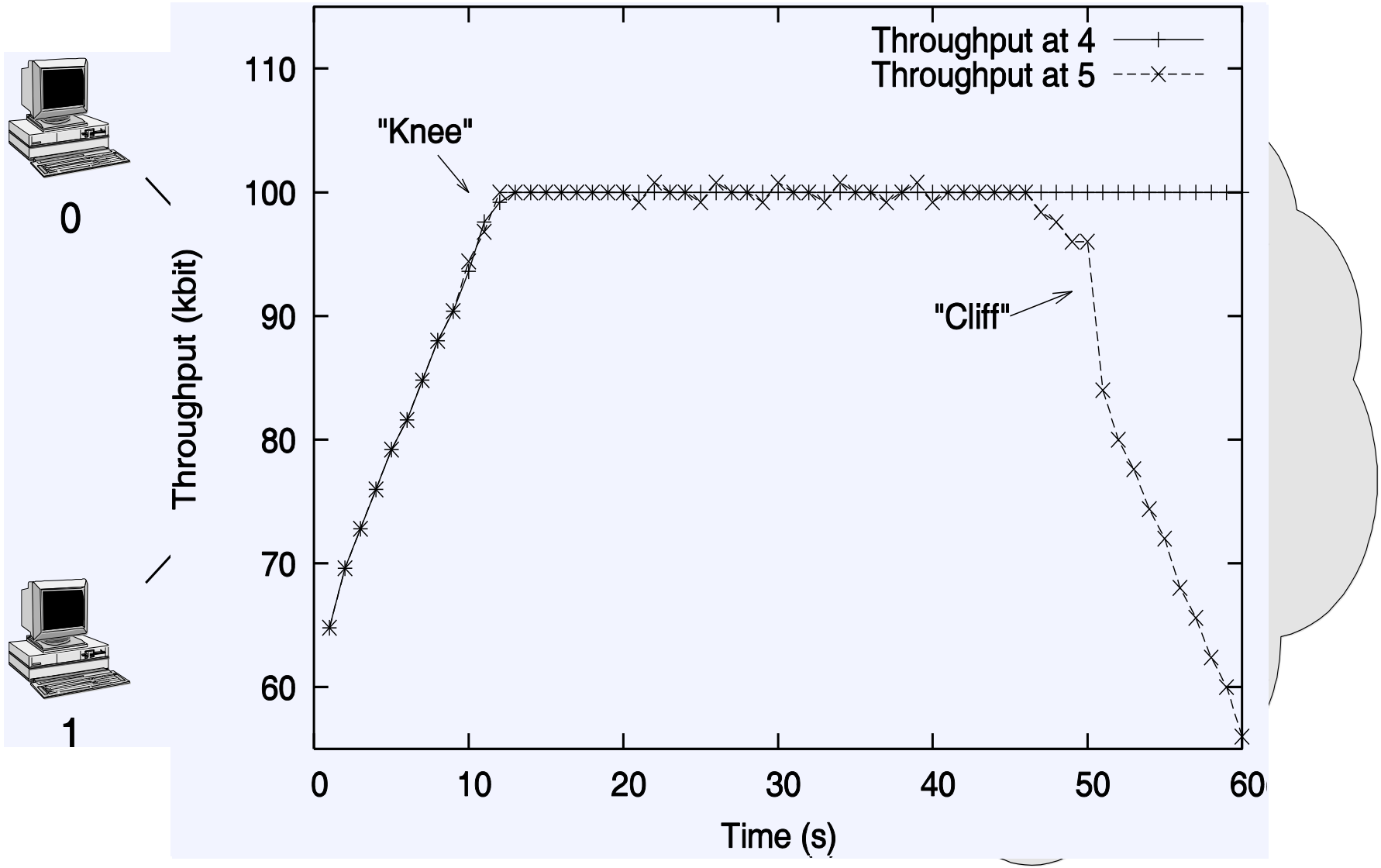
Michael Welzl

A simple router model



- Switch(ing) fabric forwards a packet (dest. addr.)
  if no special treatment necessary: “fast path“ (hardware)

- Queues grow when traffic bursts arrive
  - low delay = small queues, low jitter = no queue fluctuations
- Packets are dropped when queues overflow (“DropTail queueing“)
  - low loss ratio = small queues


0
1
Throughput (kbit)
110
100
90
80
70
60
0
10
20
30
40
50
60
Time (s)
Throughput at 4
Throughput at 5
"Knee"
"Cliff"

Global congestion collapse

Craig Partridge, Research Director for the Internet Research Department at BBN Technologies:

Bits of the network would fade in and out, but usually only for TCP. You could ping. You could get a UDP packet through. Telnet and FTP would fail after a while. And it depended on where you were going (some hosts were just fine, others flaky) and time of day (I did a lot of work on weekends in the late 1980s and the network was wonderfully free then).

Around 1pm was bad (I was on the East Coast of the US and you could tell when those pesky folks on the West Coast decided to start work...).

Another experience was that things broke in unexpected ways - we spent a lot of time making sure applications were bullet-proof against failures. (..)

Finally, I remember being startled when Van Jacobson first described how truly awful network performance was in parts of the Berkeley campus. It was far worse than I was generally seeing. In some sense, I felt we were lucky that the really bad stuff hit just where Van was there to see it.

Internet congestion control: history

- **around 1986:** first congestion collapse
- **1988:** "Congestion Avoidance and Control" (Jacobson)
  Combined congestion/flow control for TCP
  (also: variation change to RTO calculation algorithm)

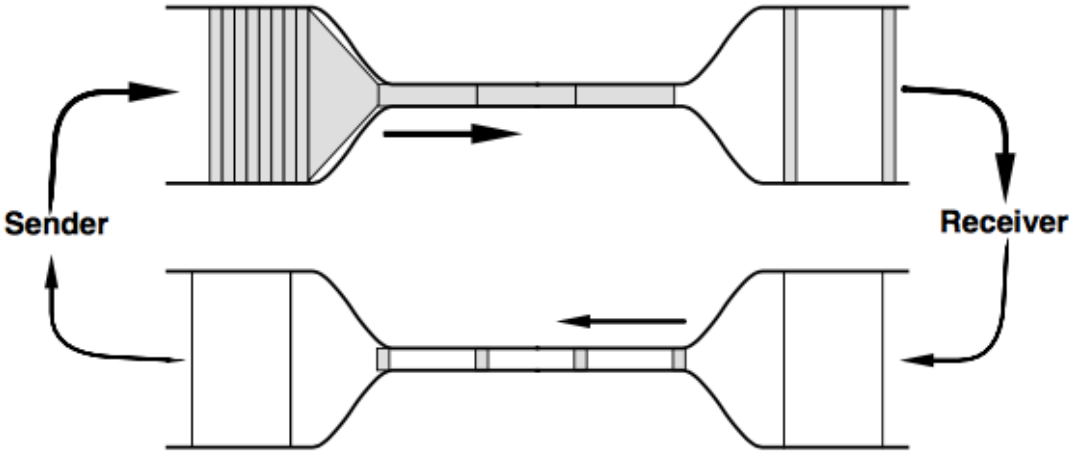
- Idea: packet loss = congestion, so throttle the rate; increase otherwise

- Goal: stability - in equilibrum, no packet is sent into the network until an old packet leaves
  - ack clocking, “conservation of packets“ principle
  - made possible through window based stop+go - behaviour

- Superposition of stable systems = stable →
  network based on TCP with congestion control = stable

A control loop to stabilize

(bring it to equilibrium and keep it there)



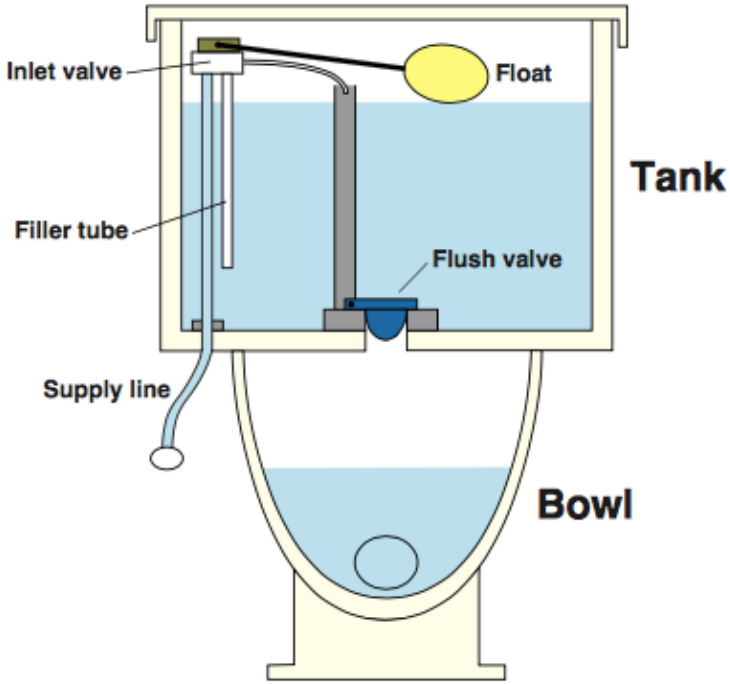
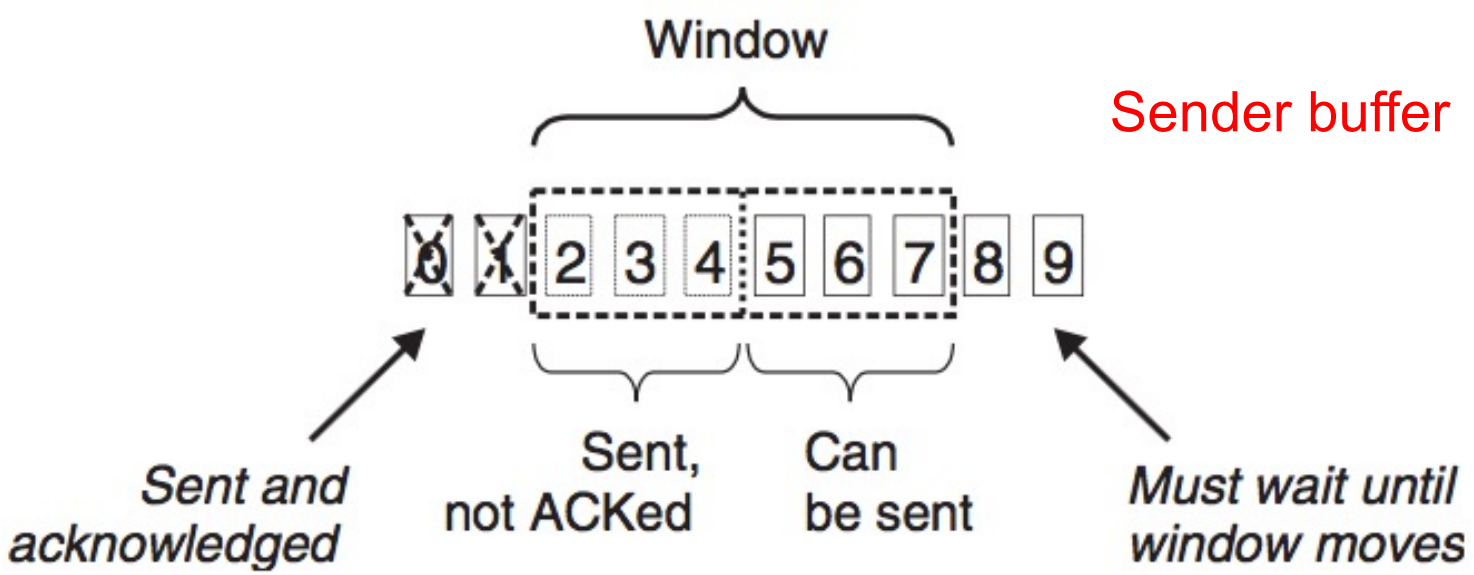


Figure 3: A well designed servo system

- Diagrams taken from:
  V. Jacobson, K. Nichols K. Poduri,
  "RED in a Different Light", unpublished
  - "...because someone on the program committee was offended by the diagram explaining its behavior."
    https://gettys.wordpress.com/2010/12/17/red-in-a-different-light/

Reminder: TCP's sliding window



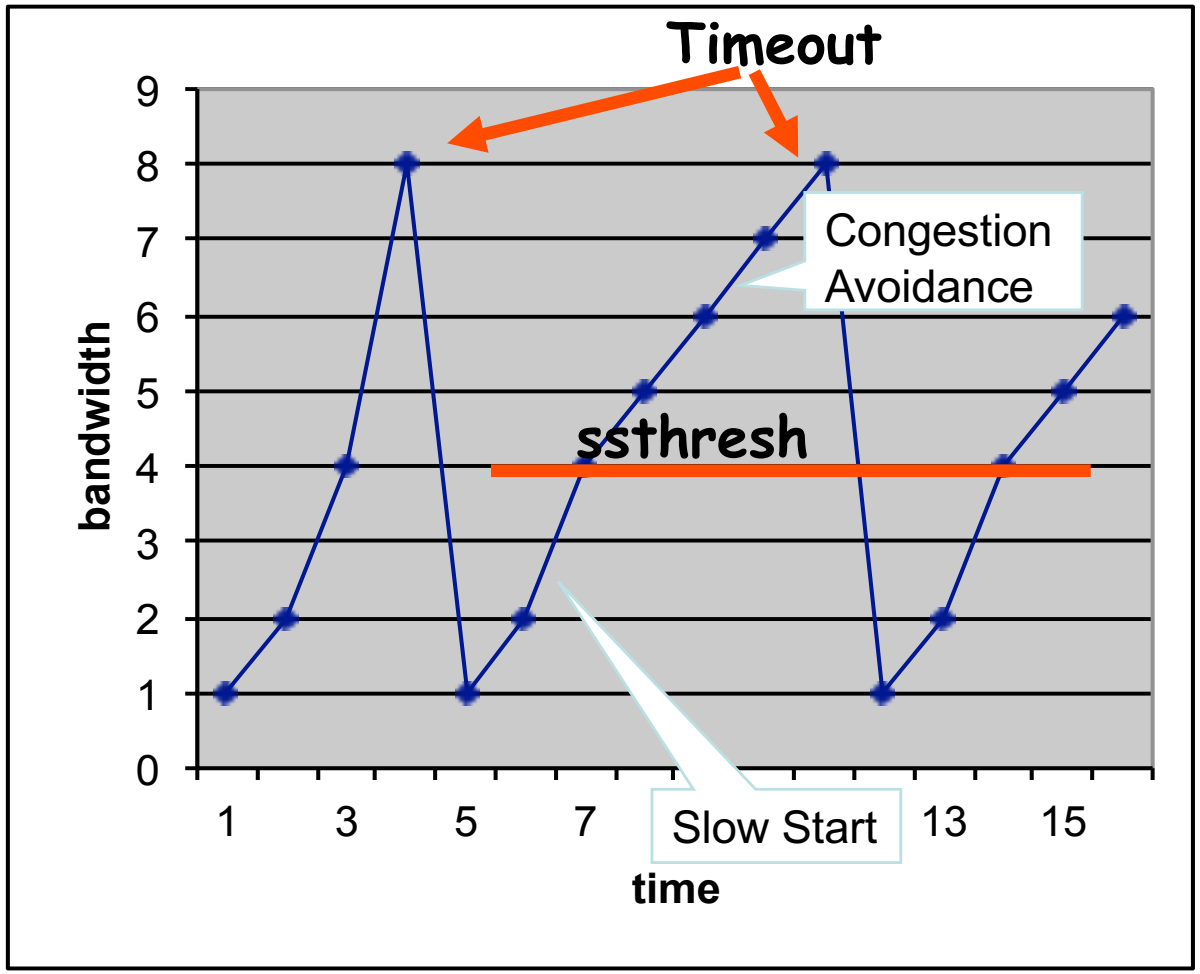
- Receiver “grants“ credit (receiver window, rwnd)
  - sender restricts sent data with window
- Receiver buffer not specified
  - i.e. receiver may buffer reordered segments (i.e. with gaps)

TCP Congestion Control: Tahoe

- Distinguish:
  - flow control: protect receiver against overload
    (receiver "grants" a certain amount of data ("receiver window" (rwnd)))
  - congestion control: protect network against overload
    ("congestion window" (cwnd) limits the rate: min(cwnd,rwnd) used!)

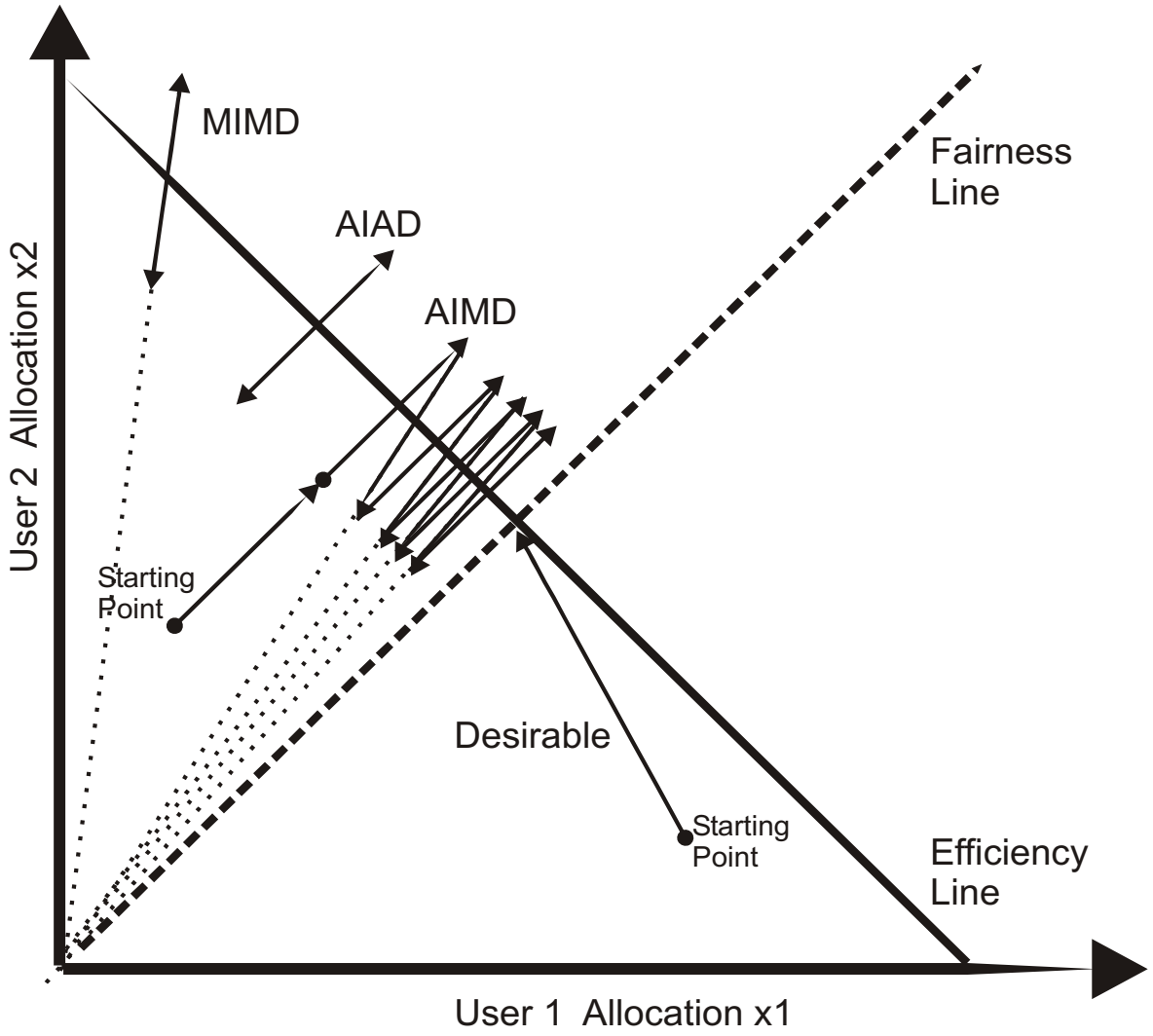
- Flow/Congestion Control combined in TCP. Two basic algorithms:
  - Slow Start: for each ack received, increase cwnd by 1 packet (exponential growth) until cwnd >= ssthresh
  - Congestion Avoidance: each RTT, increase cwnd by at most 1 packet (linear growth - "additive increase“)

TCP Congestion Control: Tahoe /2



- If a packet or ack is lost (timeout), set cwnd = 1, ssthresh = cwnd / 2 (“multiplicative decrease") - exponential backoff
  - Actually, “Flightsize/2“ instead of cwnd/2 because cwnd might not always be fully used
- Timeout based on RTT; good estimation is crucial!
- Later additions: TCP Reno, 1990
  Fast retransmit / fast recovery
  Sender detects loss via duplicate acks; no need to slow start to ssthresh

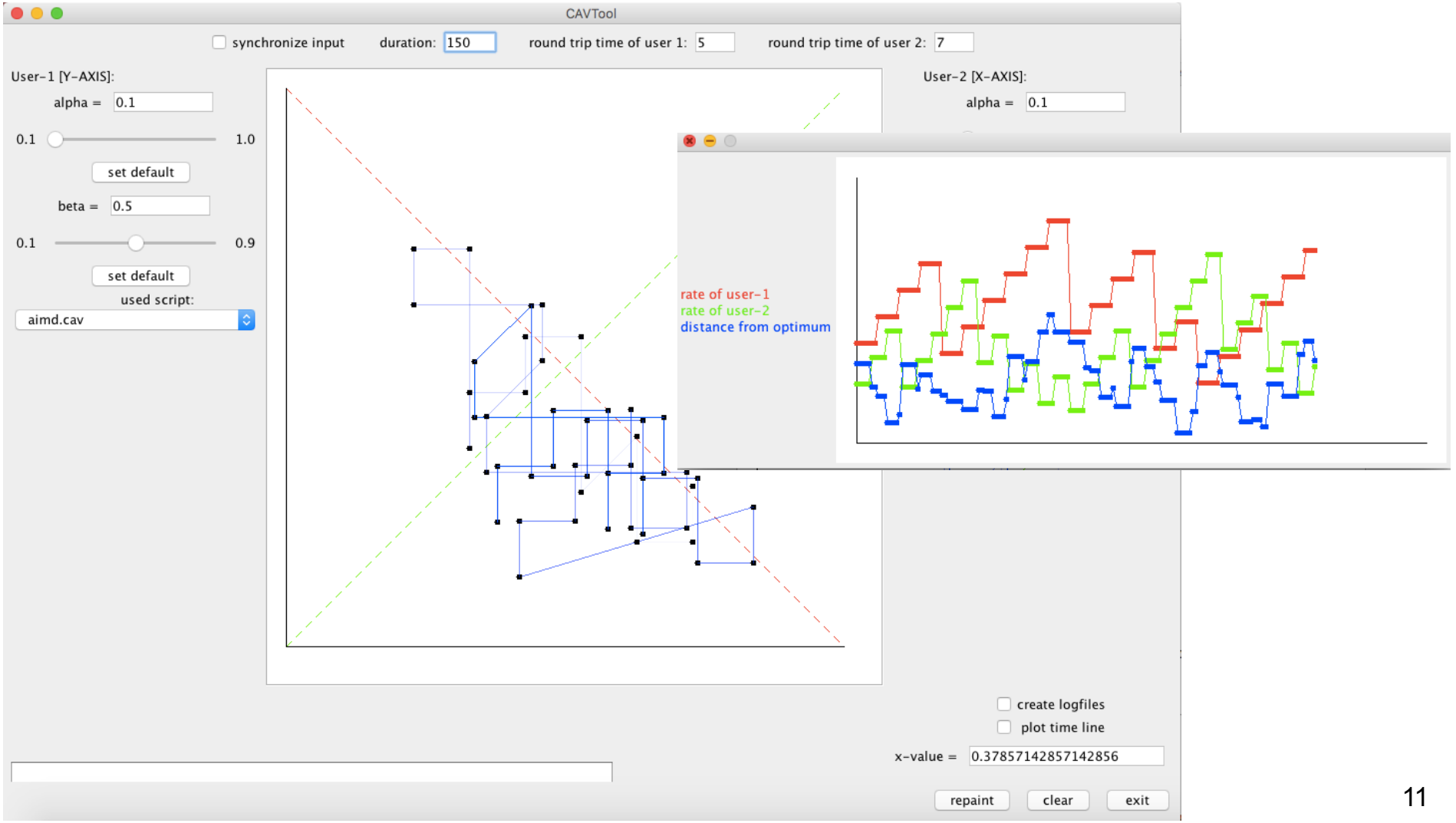
Background: AIMD



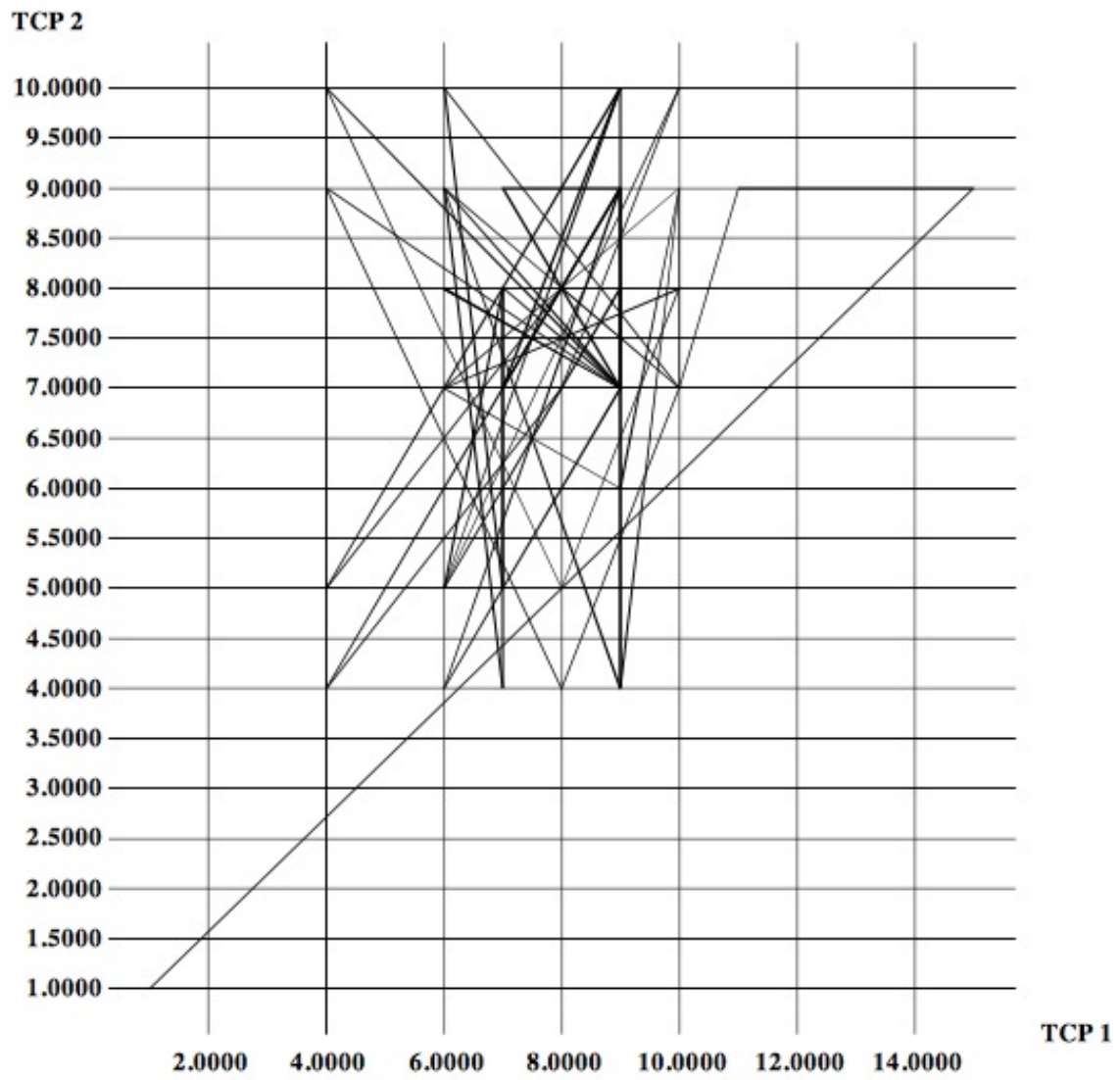
AIMD is an easy way out, but...

Michael Welzl, Max Mühlhäuser: "CAVT - A Congestion Avoidance Visualization Tool", ACM SIGCOMM CCR 3(3), July 2003, pp. 95-101.

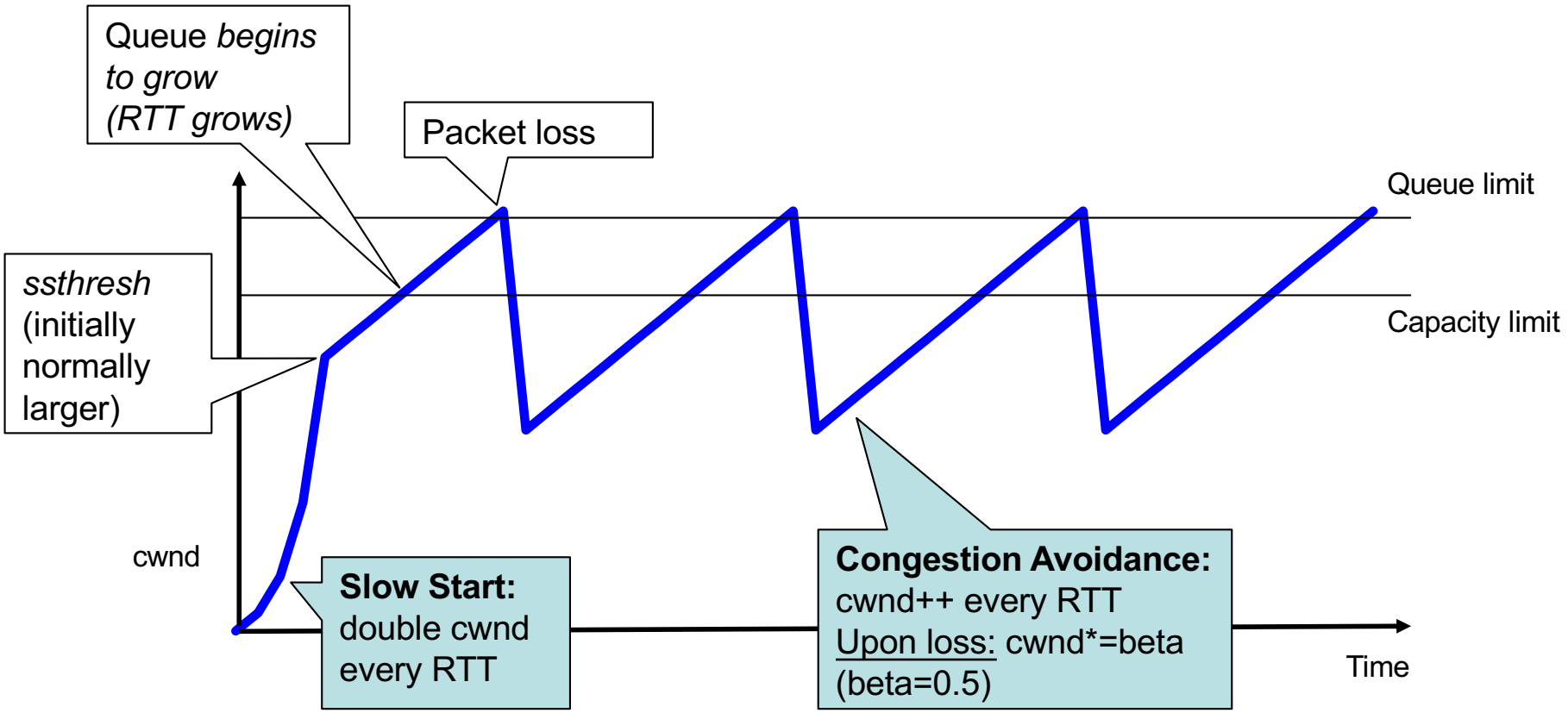
http://welzl.at/research/tools/cavt/index.html



Actual TCP (simulation)

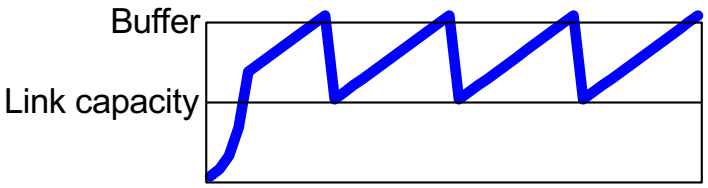


TCP Reno: building a queue



How large should a queue be?

- Old rule-of-thumb: "Bandwidth * delay product" (BDP)
  (bw = bottleneck link capacity; delay = RTT)
  - Can be unreasonably large,
    causing unnecessary delay
    (severe e.g. for VoIP and games)
  - For core routers,
    bw * RTT / sqrt(no. flows) was found to be enough
  - No new rule exists for access equipment (routers, modems, ..)

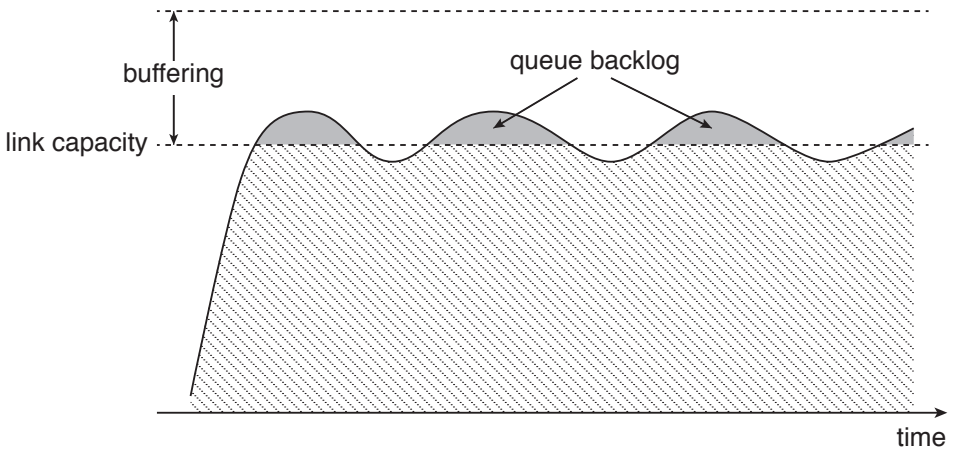


- Often, access link = bottleneck;
  access equipment's queue = the one that grows
  - Some equipment vendors sell "large buffer = feature"
  - Effect: extreme delay, now called "buffer bloat"

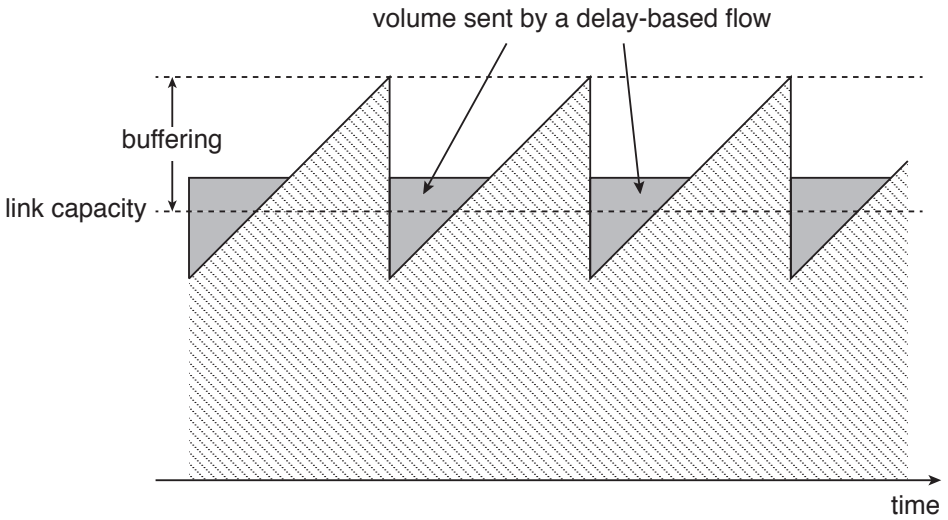
Using delay as a signal

- Delay grows before loss, so can react earlier, limit delay
  - Noisy signal, but fine-grain and easily available
  - Oldest known approach: TCP Vegas; works well but loses against Reno

- LEDBAT: turning this bug into a feature
  - For low-priority traffic that should not get in the way
  - Encapsulation not (yet?) defined; implemented over UDP in BitTorrent



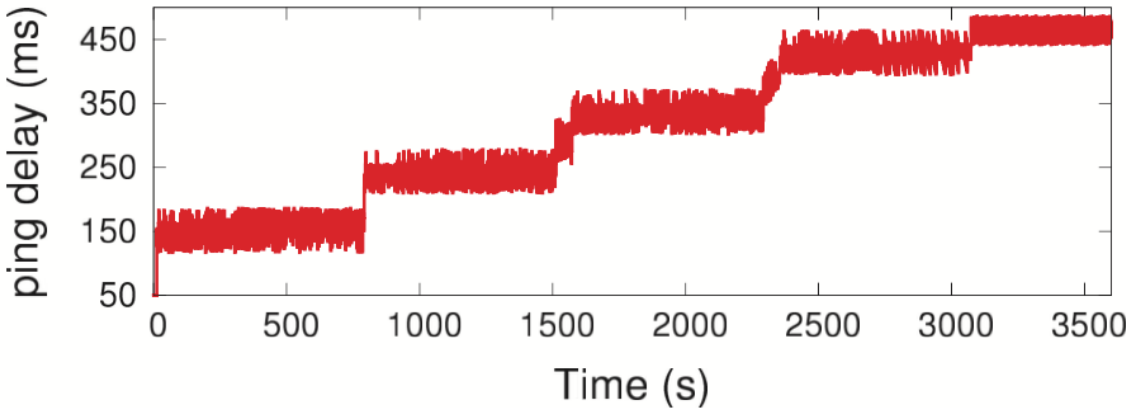
(a) Sending rate of a delay-based flow: ideal case.



(b) Coexistence of an LBE flow with a non-LBE one.

More problems with delay as a signal

- RTT easier to measure than One-Way Delay (OWD), but can be noisy (don't want to react to queue on return path)
  - Note: absolute OWD doesn't usually matter (clock sync not needed), need changes of OWD
- Size of buffer determines measurement granularity
- Base delay (long-term min. delay) can create problems
  - One flow's congestion is the next flow's base delay
  - Need to occasionally refresh, but don't measure yourself!



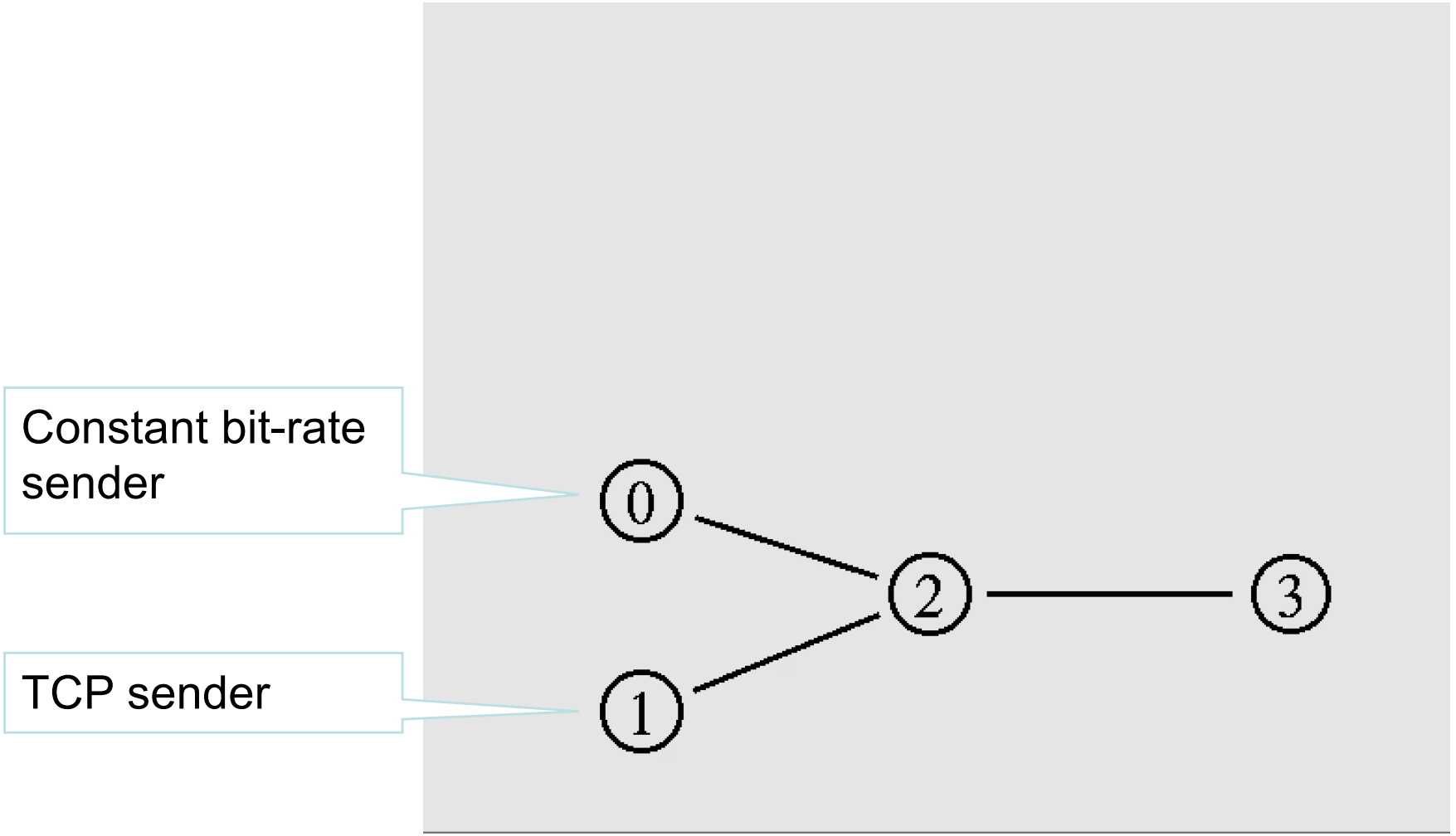
Ping delay across a 500 kbit/50ms Ethernet link, with libutp's "utp test" program running

Active Queue Management

- **1993:** Random Early Detection ("Discard", "Drop") (RED)
  (since TCP backs off as packets are dropped, drop packets earlier to *avoid* queue overflows)
  - Adds randomization to avoid traffic phase effects
  - "Punishes" unresponsive flows, improving fairness

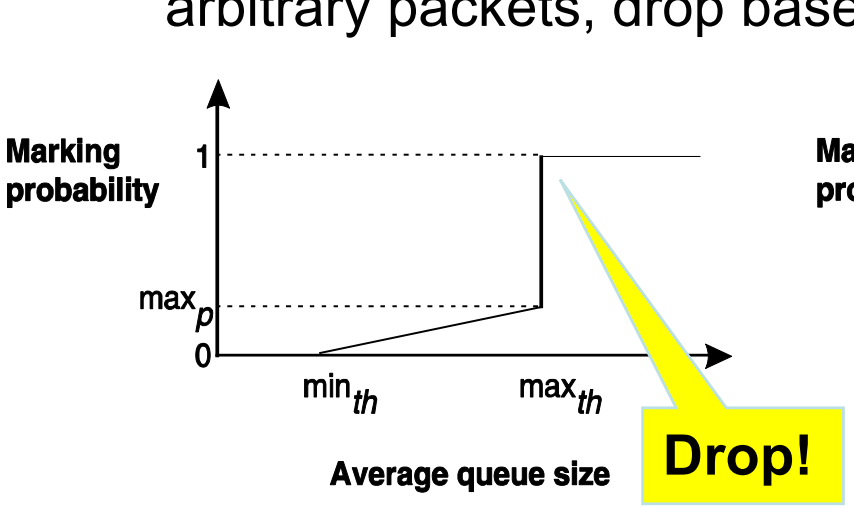
- ***Qavg = (1 - Wq) x Qavg + Qinst x Wq***
  (Qavg = average occupancy, Qinst = instantaneous occupancy, Wq = weight - hard to tune, determines how aggressive RED behaves)

Phase effects of a FIFO queue

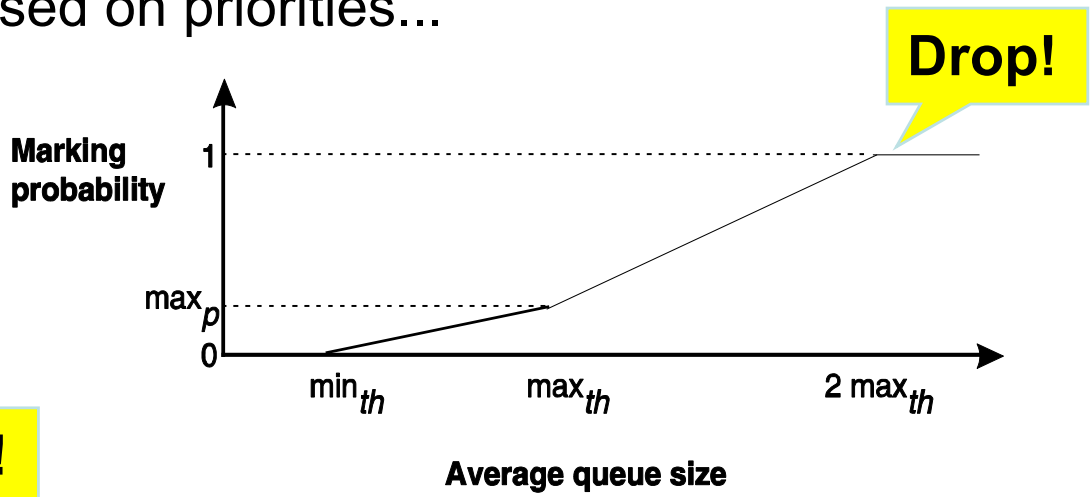


RED

- Based on exponentially weighted moving average (EWMA) of instantaneous queue occupancy = low pass filter
  - recalculated every time a packet arrives
- Several variants exist (also entirely different algorithms): drop from front, drop based on instantaneous queue occupancy, drop arbitrary packets, drop based on priorities...



RED



RED in "gentle" mode

Q: does RED interoperate well with LEDBAT?

Explicit Congestion Notification (ECN)

- **1999:** Explicit Congestion Notification (ECN)
  Instead of dropping, set a bit
  (which the receiver tells the sender about)

- End systems should act as if packet was dropped
  ⇒ real communication between hosts and routers!
  - But cannot totally replace loss reaction!

- Up to date, ECN is still hardly deployed
  - Changing now, with "Low Latency, Low Loss and Scalable Throughput" (L4S)? **Comcast** roll-out, **Apple** enabling, ...
  - Special dual-queue use of ECN bits to differentiate between "low latency" traffic and everything else
  - Richer feedback from receiver, and different send behaviour (back-off proportional to # ECN marks)

Other implicit signals: delay, arrival rate (via ACKs)

- E.g., for multimedia traffic: try to co-exist with TCP, yet avoid building a queue if possible
  - E.g., when TCP is absent

- IETF RMCAT: delay-based congestion control for WebRTC, using RTP/RTCP
  - Google Congestion Control (GCC) *(used in Chrome)*, Cisco's NADA, Ericsson's SCREAM

- BBR (also, older: TCP Westwood)
  - **RTT: only correct when below** the capacity limit
    Attainable throughput: only correct when exceeding the capacity limit
    (Sender increases the rate, but arrival rate at receiver plateaus)
  - ⇒ Need to switch between probing phases
  - First version ignored packet loss; very aggressive towards others

TCP co-existence

- Once called "being TCP-friendly"
  - i.e., do not send more than TCP would under comparable circumstances
  - Can be achieved by emulating the behaviour of TCP (or the desired parts of it)
- More modern metric: “harm“
  - Evaluates impact when sharing a bottleneck in parallel
- In practice, today, no rule is followed
  - But, e.g., when BBR was found to “eliminate“ Cubic, it was changed
  - Netflix experiments with reducing traffic bursts (video chunks to fill the buffer), instead sending more “constant“, to cause less harm

TCP's steady-state equation

- Average transmission rate T (bytes/sec):

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

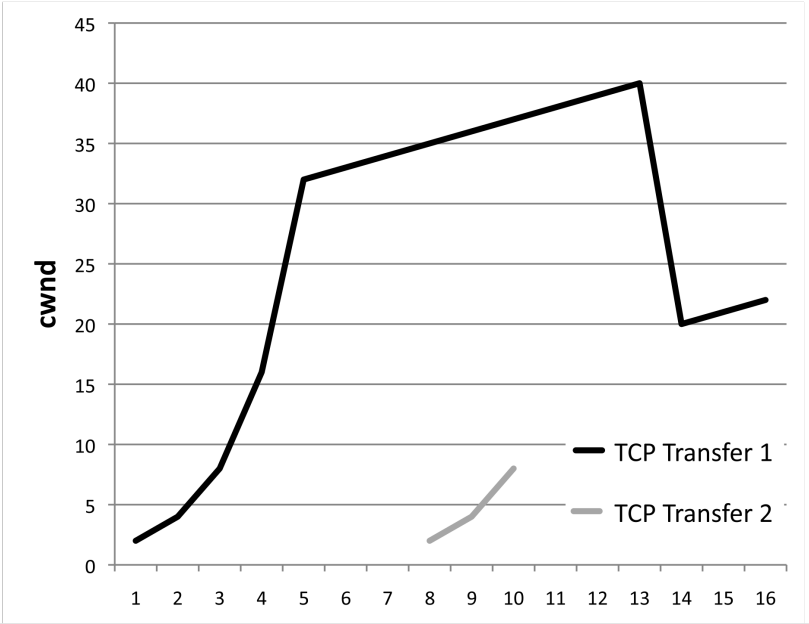
s: packet size
R: rtt
t_{RTO}: TCP retransmit timeout
p: steady-state loss event rate (the difficult part!)

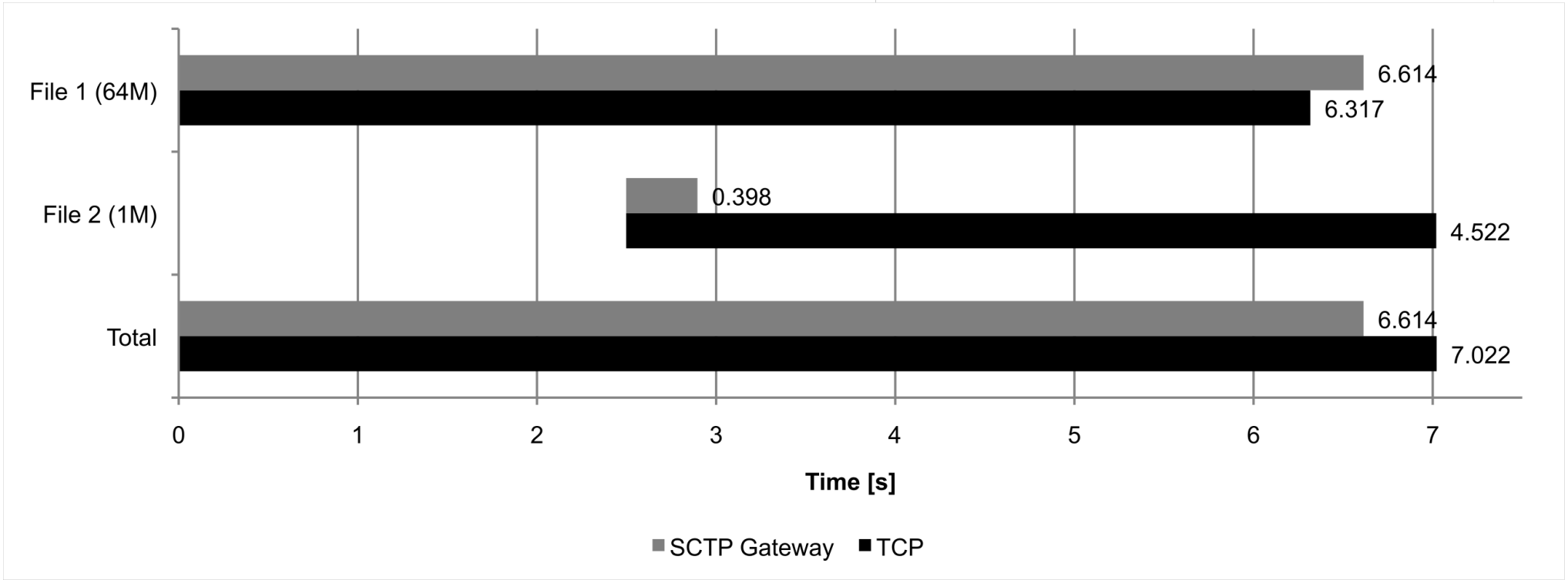
- well known old mechanism TFRC (TCP-friendly rate control) used this as an upper bound, but with a smoother sending rate than TCP
  - Less delay fluctuation, good for multimedia
- GCC also uses this equation, but as a lower limit

Congestion state and multiple flows

- TCP Control Block (TCB): data such as RTO, cwnd, ..
  - Related to path, yet separately stored per TCP connection
  - Compare: layering problem of PMTU storage
- Multiple TCP connections are not TCP-friendly to one
  - Neither are short flows which start and stop in Slow Start
  - Web browsers open hundreds!
- Yet, sometimes fewer competing flows can be better
  - E.g., new short flow can use already-large cwnd
  - HTTP/2 replaced "sharding", now HTTP/3 (= HTTP / QUIC / UDP)
- TCB interdependence: share TCB info when flows start
  - Temporal sharing: learn from previous connection
  - Ensemble sharing: learn from existing connections

Combining congestion control: an example





MPTCP: being below-TCP-friendly

- Many modern hosts are multihomed
  - Smartphones (WiFi + 3G), data centers
  - Why not use both connections at once?



- Cannot know where bottleneck is
  - If it is shared by the two connections, they should appear (be as aggressive) as only one connection
  - MPTCP changes congestion avoidance “increase” parameter to “divide” aggression accordingly
    - but instead of being “½ TCP”, tries to send as much as possible over least congested path
    - Least congested = largest window achieved; hence, increase in proportion to window

Fairness principles

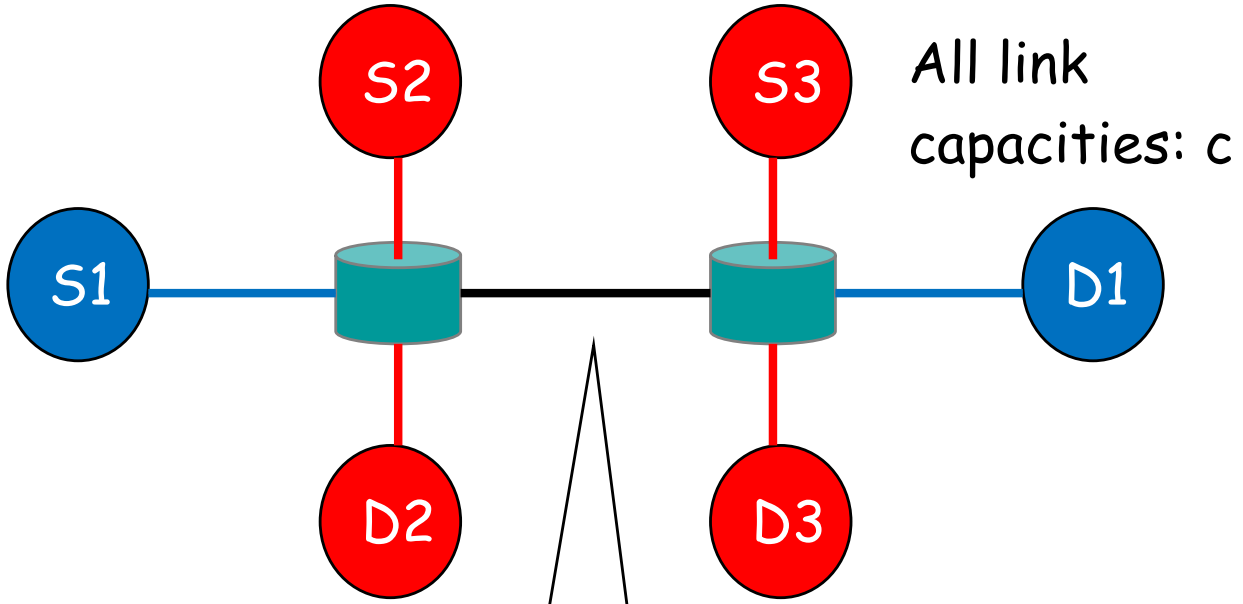
- TCP-friendliness is a fairness definition
  - Based on technical behavior of TCP... is this meaningful?

- E.g., ATM ABR: Max-Min-fairness
  - "A (..) allocation of rates is max-min fair iff an increase of any rate (..) must be at the cost of a decrease of some already smaller rate."
  - One resource: mathematical definition satisfies "general" understanding of fairness - resource is divided equally among competitors
  - *Usually* requires knowledge of flows in routers (switches) - scalability problem!

Proportional Fairness

F. Kelly: Network should solve a global optimization problem (maximize log utility function)

Max-Min-fairness suboptimal:
S1 = S2 = S3 = c/2



Proportionally fair allocation:
S1 = c/3, S2 = S3 = 2c/3

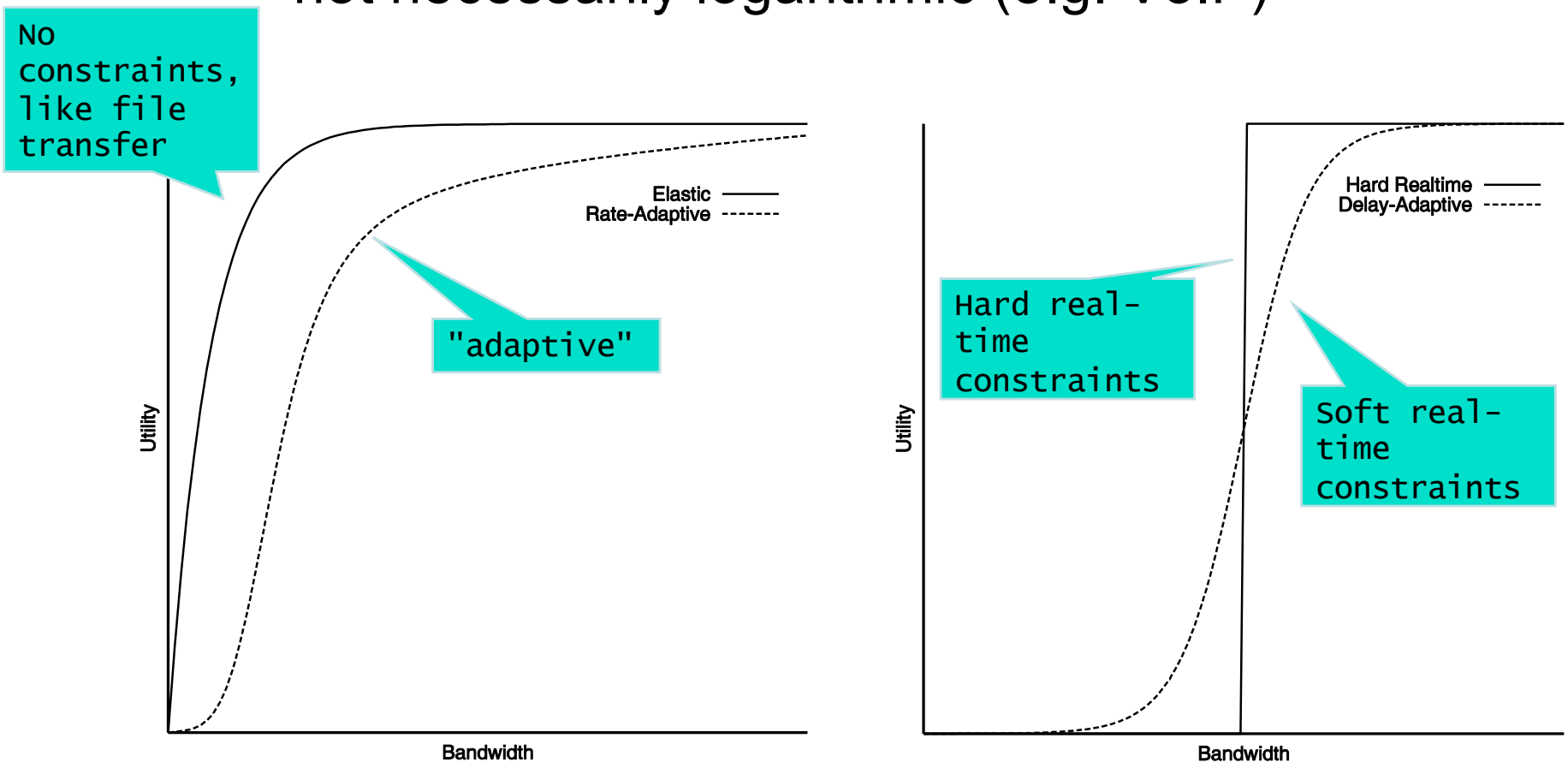
Proportional fairness:

"An allocation of rates $\vec{x}$ is proportionally fair iff, for any other (..) allocation $\vec{y}$, we have: $\sum_{s=1}^{S} \frac{y_s - x_s}{x_s} <= 0$ "

*(**roughly** approximated by AIMD!)*

Utility functions

- Consider: multimedia applications
  - not necessarily logarithmic (e.g. VoIP)



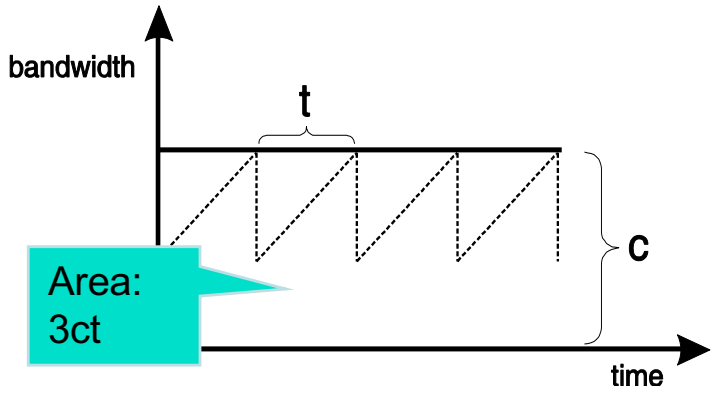
TCP with High-Speed links

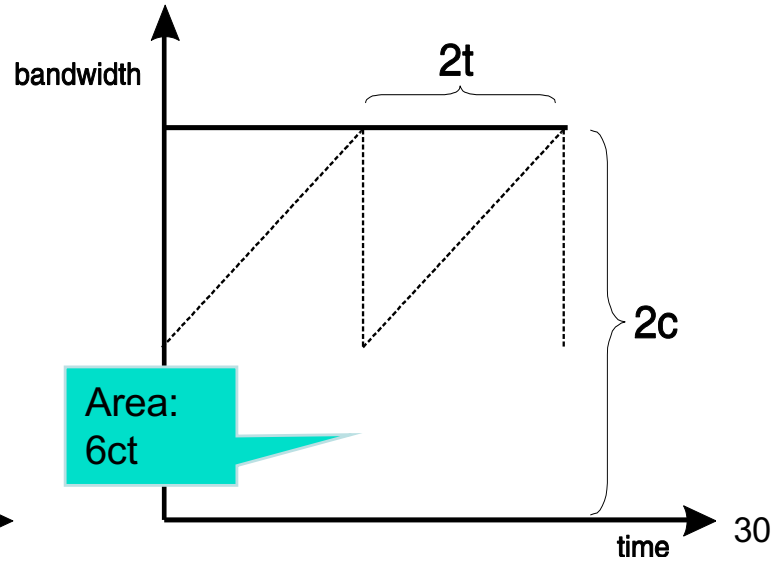
- TCP over “long fat pipes“: large bandwidth*delay product
  - long time to reach equilibrium, MD = problematic
  - from RFC 3649 (HighSpeed RFC, Experimental):

```
For example, for a Standard TCP connection with 1500-byte packets and a
100 ms round-trip time, achieving a steady-state throughput of 10 Gbps
would require an average congestion window of 83,333 segments, and a
packet drop rate of at most one congestion event every 5,000,000,000
packets (or equivalently, at most one congestion event every 1 2/3
hours). This is widely acknowledged as an unrealistic constraint.
```

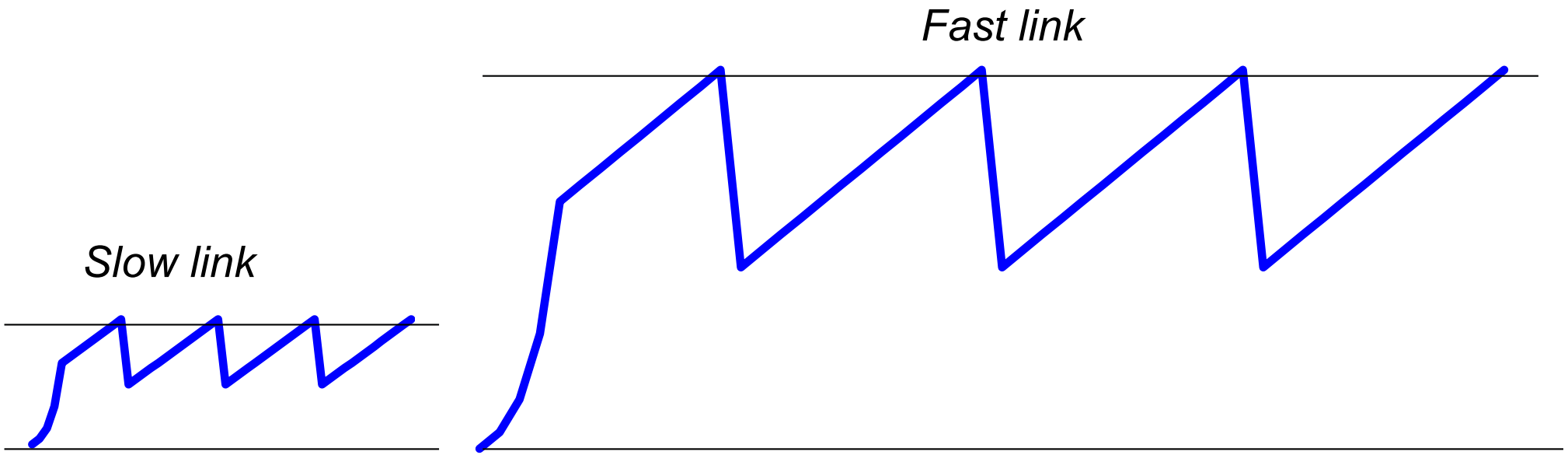
Theoretically, utilization independent of capacity

But: longer convergence time



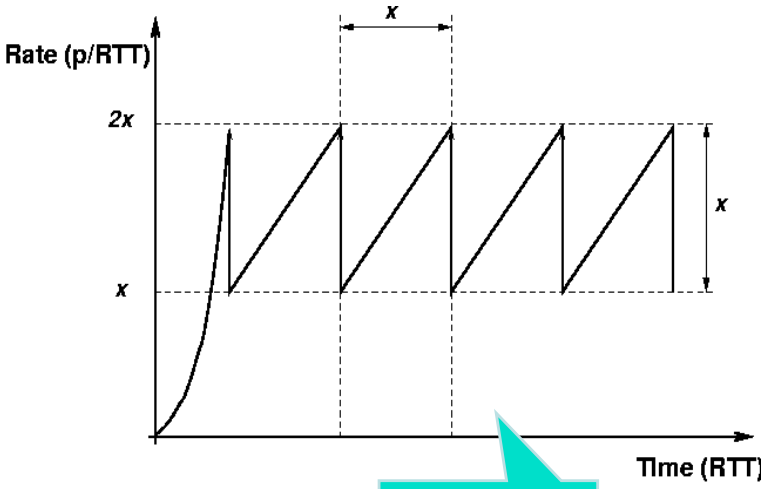


Slow convergence animation

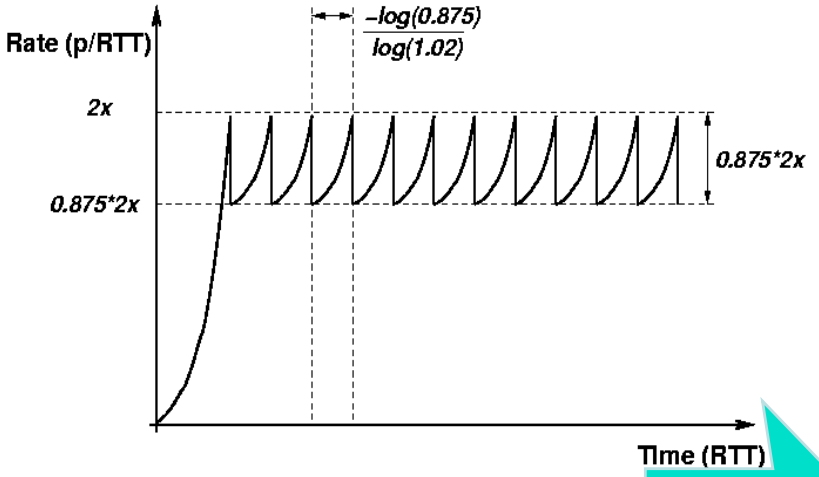


Some proposed solutions

- Scalable TCP: increase/decrease functions changed
  - cwnd := cwnd + 0.01 for each ack received while not in loss recovery
  - cwnd := 0.875 * cwnd on each loss event
    (probing times proportional to rtt but not to the capacity)



Standard TCP



Scalabe TCP

Source: http://www.deneholme.net/tom/scalable/

Some proposed solutions /2

Rate	Standard TCP recovery time	Scalable TCP recovery time
1Mbps	1.7s	2.7s
10Mbps	17s	2.7s
100Mbps	2mins	2.7s
1Gbps	28mins	2.7s
10Gbps	4hrs 43mins	2.7s

- HighSpeed TCP (RFC 3649, includes Scalable TCP discussion):
  - response function includes a(cwnd) and b(cwnd), also depend on loss ratio
  - less drastic in high bandwidth environments with little loss ***only***
  - **Significant step!**
  - Previously, either TCP-friendly or better-than-TCP; no combinations!

- "TCP-friendly" is now a term of the past
  - *"There is no point in being friendly to TCP, it will not return the favour".* [Andrew McGregor]

Proposed solutions /3

- BIC, CUBIC
  - BIC (Binary InCrease TCP) uses binary search to find the ideal window size:
    - when loss occurs, current window = max, new window = min
    - check midpoint;
    - if no loss ⇒ new min, increase; else new window = new max
  - CUBIC = BIC++ using **cubic function**

- After major press release (Slashdot: "BIC-TCP 6000 times quicker than DSL"), BIC became the default TCP CC. in Linux in mid-2004!
  - Major change to the CC. landscape
  - Soon after, replaced with **CUBIC**



government technology

PURESTORAGE

EVALUA
STORAG

HARDWARE AND SOFTWARE

North Carolina Scientists Develop Breakthrough Internet Protocol

BIC-TCP can achieve speeds roughly 6,000 times faster than DSL

BY NEWS REPORT / MARCH 16, 2004

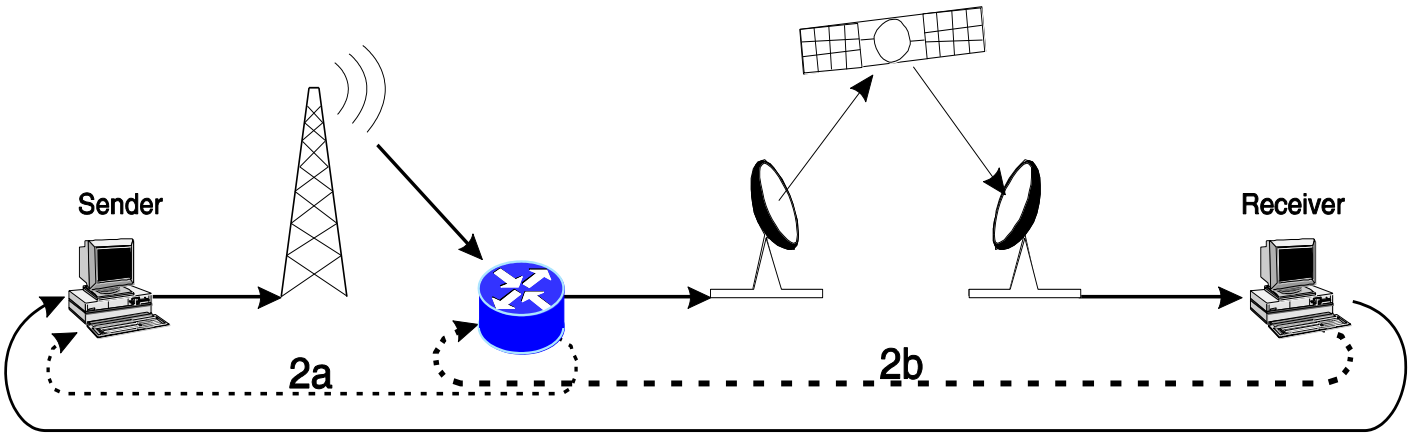
Tuesday, March 16th, 2004

NEW INTERNET PROTOCOL BLAZING FAST

N.C. State researchers presented a paper in Hong Kong yesterday about a new protocol they developed that could send TCP into the history books. The new protocol, BIC (Binary Increase Congestion)-TCP, is 6,000 times faster than any of today's DSL services

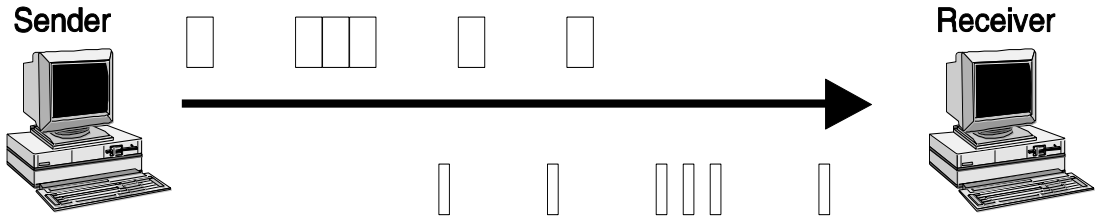
TCP over Satellite, and PEPs

- Satellites combine several problems
  - Long delay + high capacity (= large BDP)
  - Wireless (but usually not noisy (for TCP) because of link layer FEC)
  - Can be asymmetric (e.g. direct satellite downlink, 56k modem uplink)
- Thus, TCP needs help over satellite
  - Transparent improvements ("Performance Enhancing Proxies") common
  - Figure: split connection approach: 2a / 2b instead of control loop 1
  - Many possibilities - e.g. Snoop TCP: monitor + buffer; in case of loss, suppress DupACKs and retransmit from local buffer
  - But: today, this only works with TCP (e.g., not the all-encrypted QUIC protocol)

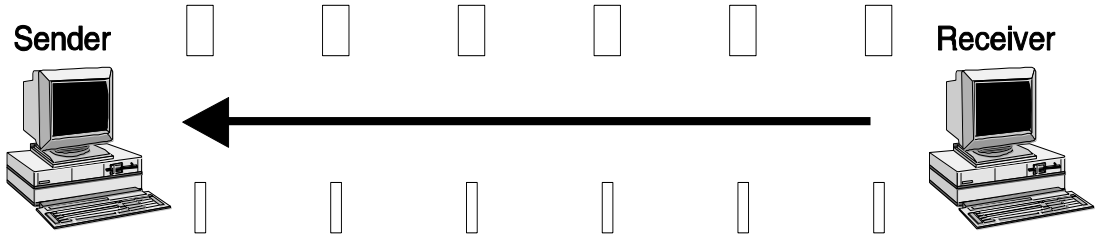


Pacing: increasingly common

- "Micro burstiness" can lead to delay + packet drops
- Generally, packet gap dictated by bottleneck link; but incoming data can be bursty (e.g. from slow start)
- Pacing is hard at high speeds (clock granularity)
  - But getting more feasible; increasingly popular!
  - Common solution is to allow very small bursts (e.g. 3-4 packets)



(i) without pacing



(ii) with pacing

Congestion control reality check
(note: no guarantee for correctness of this slide!)

- Linux: Cubic (default, many others available)
  - Cubic comes with a Slow Start early exit based on delay heuristics, called HyStart
- Apple: Cubic? and MPTCP
  - LEDBAT for OS updates
  - Hosts sometimes try to negotiate ECN (*someone* needs to make a start!)
- Windows: Cubic default for Windows 10, Windows Server 2019
  - But with their own new HyStart++
  - Old delay+loss variant Compound-TCP (CTCP) was default TCP CC. in Windows Server 2008 beta for testing purposes
  - When it sees tiny RTTs, switches to Datacenter TCP (DCTCP), a very aggressive ECN based variant ("TCP Prague", for L4S, is based on this)
  - LEDBAT++ for OS updates

Messy...
but the Internet still works ☺

- **Applications:**
  - BitTorrent: LEDBAT
  - WebRTC: RMCAT congestion controls
  - YouTube: Google experiments with their own BBR cc. (now version 3)
    (Netflix uses Reno, but experiments with a BBR variant, in-house)
  - Facebook: Meta experiments with delay-based COPA cc. for video

Alexa top 20k websites 2009: No more Reno!

Variant	Websites	Proportion
CUBIC [15]	6,139	30.70%
BBR [4]	3,550	17.75%
BBR G1.1	167	0.84%
YeAH [2]	1,162	5.81%
CTCP [34]/Illinois[22]	1,148	5.74%
Vegas [3]/Veno [13]	564	2.82%
HTCP [21]	560	2.80%
BIC [37]	181	0.90%
New Reno [28]/HSTCP [12]	160	0.80%
Scalable [20]	39	0.20%
Westwood [7]	0	0.00%
Unknown	3,535	17.67%
Short flows	1,493	7.46%
Unresponsive websites	1,302	6.51%
Total	20,000	100%

„New“ only refers to loss recovery (next week!), but „New Reno“ is the Linux name

⇒ wrong, yet common to use this to refer to Reno CC.

Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. 2019. The Great Internet TCP Congestion Control Census. Proc. ACM Meas. Anal. Comput. Syst. 3, 3, Article 45 (December 2019), 24 pages.

Alexa top 20k websites 2024: Less BBR, Cubic still strong

- From different viewpoints

Variant	Ohio		Paris		Mumbai		Singapore		Sao Paulo	
	Websites	Share	Websites	Share	Websites	Share	Websites	Share	Websites	Share
BBRv1	2,594	13%	1,900	9.5%	1,635	8%	2,541	12.7%	1,280	6.4%
BBRv2	515	2.6%	373	1.9%	12	0.1%	251	1.3%	0	0%
BIC	712	3.5%	807	4%	837	4.2%	402	2%	227	1.1%
CUBIC	8,202	41%	8,406	42%	8,822	44.1	8,673	43.4%	6,982	34.9%
HSTCP	0	0%	0	0%	0	0%	0	0%	0	0%
HTCP	583	2.9%	370	1.9%	522	2.6%	421	2.1%	363	1.8%
Illinois	721	3.6%	684	3.4%	1,121	5.6%	625	3.1%	229	1.1%
New Reno	1,840	9.2%	1,509	7.5%	3,032	15.2%	2,093	10.5%	2,683	13.5%
Vegas	878	4.4%	421	2.1%	301	1.5%	526	2.6%	511	2.5%
Veno	112	0.6%	382	1.9%	52	0.3%	21	0.1%	11	0.1%
Westwood	201	1%	170	0.9%	0	0%	0	0%	0	0%
Scalable	18	0.1%	0	0%	0	0%	6	0%	0	0%
YeAh	123	0.6%	89	0.4%	64	0.3%	0	0%	112	0.6%
Unknown	3,501	17.5%	4,889	24.4%	3,602	18.1%	4,441	22.2%	7,602	38%
Unresponsive	0	0%	0	0%	0	0%	0	0%	0	0%
Total	20,000	100%	20,000	100%	20,000	100%	20,000	100%	20,000	100%

Ayush Mishra, Lakshay Rastogi, Raj Joshi, "Keeping an Eye on Congestion Control in the Wild with Nebby", ACM SIGCOMM 2024, August 2024.

Thank you!

A humble reference suggestion:

