

Extending TCP for Low Round Trip Delay

Logarithmically Scaled Additive Increase Multiplicative Decrease (LS-AIMD)

Asad Sajjad Ahmed^{1,2}
asadsa@ifi.uio.no

¹Department of Informatics
Faculty of mathematics and natural sciences
UNIVERSITY OF OSLO

²Department of Advanced Computing and System Performance (CASPER)
SIMULA RESEARCH LABORATORY

Master thesis, Autumn 2019

This thesis would not have been possible without:

- Bob Briscoe (main supervisor)

Independent, Simula

- For his tremendous help in understanding, problem solving, design document and scientific writing.

- Andreas Petlund, Carsten Griwodz & Håkon Kvale Stensland (internal supervisors)

Simula, University of Oslo (UiO)

- For their great help in making a submission of this thesis possible.

This thesis would not have been possible without:

- Joakim Misund

University of Oslo (UiO), Simula

- For help in troubleshooting packet loss during flow start-up under Linux TCP with ECN and for his comments on AI in CWR rounds.

- Matt Mathis & Yuchung Cheng

Google

- For their inputs in troubleshooting a regression bug in newer versions of the Linux kernel.

This thesis would not have been possible without:

- Family & friends
 - For their tremendous support throughout the thesis.

However, any opinion expressed herein remains the word of the speaker and no one else, unless otherwise stated.

Why care about low latency first now?

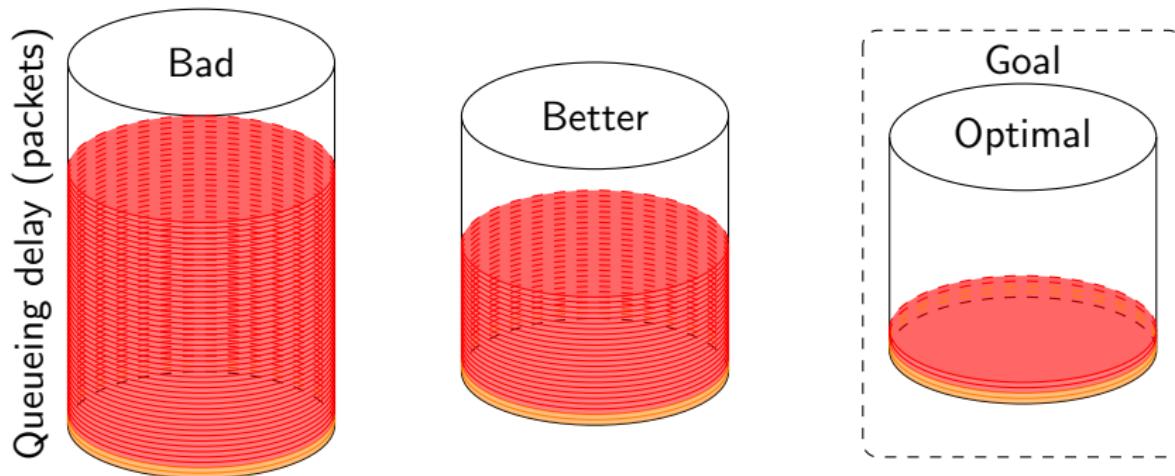
- Legacy applications: gaming, VoIP
 - Modern applications: web, instant messaging, virtual/augmented reality, cloud gaming, video conferencing/streaming, emergency systems, remote assistance, drones, etc.
 - Data Centres (DCs) & Content Delivery Networks (CDNs)
 - Inter Process Communication (IPC) within isolated environments
 - Not only the Internet, but anything which involves any form of greedy resource sharing over a common bottleneck, i.e. a queue.

So what is the applications latency typically made out of and where to cut?

- Propagation delay - Time taken for 1-bit to travel from the source to destination.
$$\text{distance} / \text{speed of the medium}$$
 - Transmission delay - The serialisation time for a packet.
$$\text{frame size} / \text{link speed}$$
 - Queueing delay - Time spent waiting for transmission behind other packets. Takes at least one transmission delay to dequeue one packet from the head of the queue.
 - Processing delay - Cost to process the packet in terms of computational power.

How to cut the queueing delay?

- Active Queue Managements (AQM)
 - Explicit Congestion Notification (ECN)
 - Shallow marking threshold over a lower quantity of buffer.

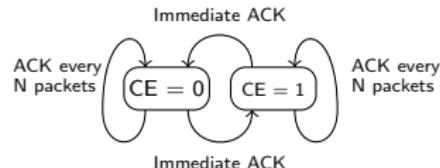


The evolution of AQMs

How to maintain a tiny queue?

Take in use a scalable congestion controller, e.g. DCTCP, TCP Prague, BBR v2.

- Keep the AQM simple; Neither smooth the queue or use any burst before yielding a mark.
- Use an improved ECN feedback loop at the receiver to feed the sender about the extent of congestion.
- Upon arrival of the first mark within a RTT use the calculated estimate, α , to perform a more gradual reduction.



$$\alpha = \alpha * (1 - g) + g * F, \quad 0 \leq \alpha \leq 1$$

$$S_s = \max(inflight_s * (1 - \alpha/2), \quad 2s)$$

Figure: SM: DCTCP ECN ACK
 (Figure 10 of the DCTCP paper)

How to incrementally deploy this over the Internet?¹

- Use Dual Queue AQM to segregate the new traffic from the legacy TCP traffic.
- Use the last ECN codepoint, ECT(1), to classify scalable senders.
- Provide isolation for low latency, but keep capacity sharing of the same level.

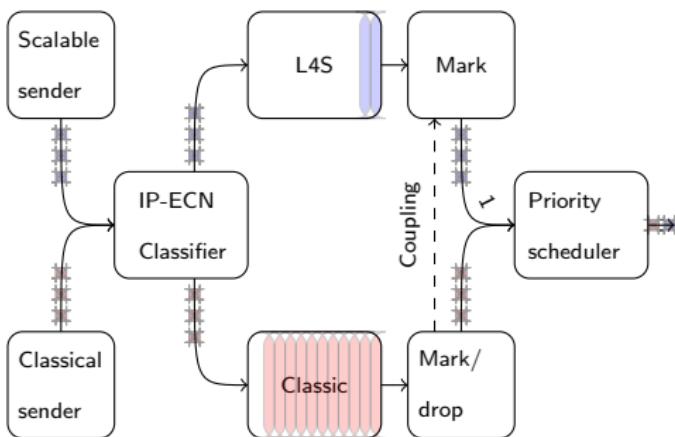


Figure: L4S Architecture [2]

¹Work in progress

The submss regime

Then what is the problem?

Two very hard and fundamental scalability issues of TCP. Not easy to solve as they both conflicts with the traditional way to perform congestion control.

- Minimum transmission rate of TCP² per RTT cause unresponsiveness when probing over ECN enabled network.
- Constant additive increase of TCP scales for neither low or large congestion window values.

²In theory, TCP cannot become unresponsive when following RFC3168/RFC5681, but an implementation which tries to stay responsive yield poor performance under low BDP paths. Nonetheless, Linux TCP is good example which selectively strips this requirement.

But does not a higher quantity of bitrate mean a higher transmission rate per RTT?

$$BDP[b] = Bandwidth[bps] * RTT[s]$$

$$fair\ rate[b] = BDP[b] / senders[\#]$$

$$W_s = BDP[b] / (senders[\#] * ethernet_frame[b])$$

Not always given:

- What if the base RTT is of a significantly lower quantity as well?
 - What about a higher quantity of competitors at the bottleneck?
 - What does a higher MTU in the network mean for TCP?

The submss regime

Why not let the AQM do packet eviction instead?

- The AQM should always evict packets to hinder any unresponsive sender from building a queue.
- Thus, prevent any malicious/malfunctioning sender from causing higher RTT for other traffic.
- However, an AQMs which performs evictions should not be a standalone solution to the problem, but instead be a *consequence* for any sender who decides to challenge the AQM by not falling back.
- Reasons: wastes the limited resources of the network, induces extremely high packet loss rate, harder to perform any efficient loss recovery.

An overview of the problem

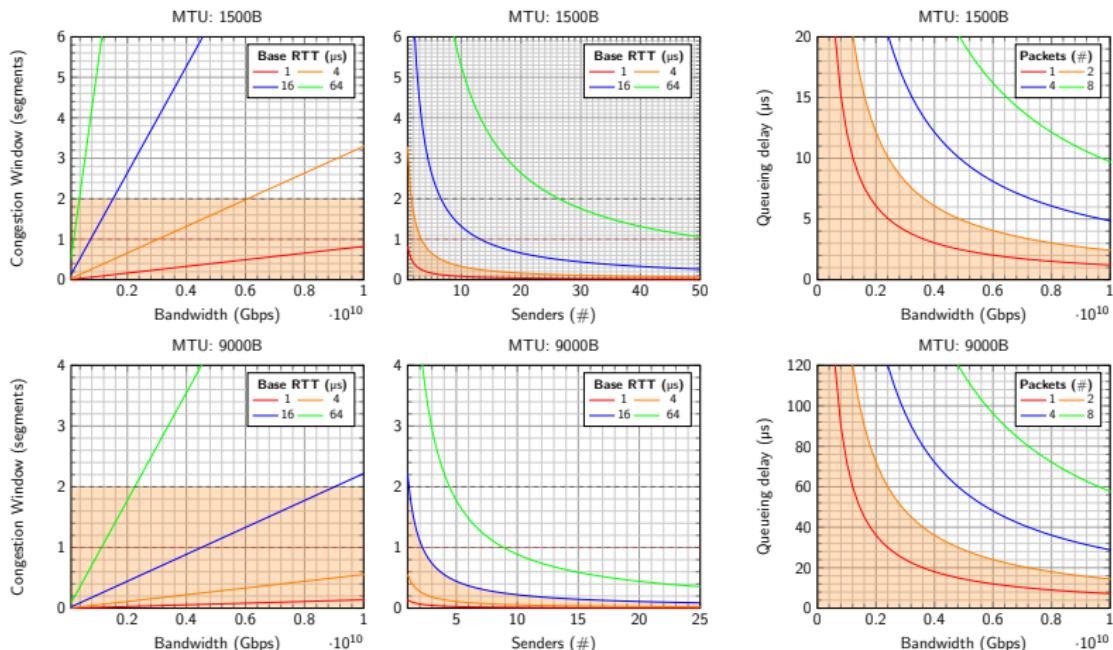


Figure: Plot of congestion window sizes and queueing delay for upto 10 Gbps

Asad Sajjad Ahmed (supervised by: Bob Briscoe)

IFI, Simula

Our research questions

- Can we extend TCP for shallow base RTT?
 - Can we preserve stretch acknowledgements for low congestion window values?
 - Can we preserve a congestion window made out of segments in a network with a low transmission rate per RTT?
 - Are we able to create a minimal solution which is easy to implement?

Limitations

Limitations in our research

- We do not evaluate TCP under any level of packet loss. Our research instead rely on ECN as a signal for congestion. However, our work does not ignore packet loss per se, but is underprioritised.
- Develop and test our prototype under a stable version of the Linux kernel.
- Congestion control algorithms: TCP Reno & DCTCP.
- Experiments conducted over a simple broadband topology.

How has the problem evolved in the research community?

- The problem has initially been a limiting factor for many concurrent incast flow for TCP, where a higher quantity of bitrate is needed to scale for a greater number of competitors through the bottleneck.
- However, recent work at removing the deep queue of the bottleneck both in data centres and L4S over broadband testbed has rediscovered the same scalability problem of TCP, but now due to a lower base RTT because of a tiny queue.
- The problem is now amplified and cannot be solved with more bitrate. The problem makes TCP unable to keep up with a low queue AQM.

What are the main contributions from others for the problem we have in front of us?

- 97 - TCP - Morris - Many flows [17]
 - 97/99 - TCP - Feng et al. - Adaptive RED & SUBTCP [7, 8]
 - 97 - TCP - Floyd et al. - ECN implementation in ns [9]
 - 01 - TCP - Qiu et al. - Performance evaluation of many flows [19]
 - 01/03 - TCP - Hsiao et al. - Delay control [10, 11, 15, 18]
 - 02 - TCP - Venkataramani et al. - TCP Nice [20]
-

- 11 - TCP - Chen et al. - The Sub-packet regime & TAQ [3–5]
- 14 - LEDBAT - Komnios et al. - Sub-packet regime [14]
- 15 - DCTCP - Miao et al. - DCTCP+ [16]
- 15/18 - DCTCP - Huang et al. - Packet slicing [12, 13]
- 17 - DCTCP - Cho et al. - ExpressPass [6]

Experiment plan

How to test TCP?

Concrete Plan

Experiment (#)	1	2	3A	3B	4A	4B	4C	4D	4E	4F	4G	4H
Bitrate (Mbps)	200	200	200	200	600	600	600	600	600	600	600	600
Base RTT (us)	300	300	100	100	250	500	750	1000	250	500	750	1000
MTU (B)	1500	1500	1500	1500	1500	1500	1500	1500	9000	9000	9000	9000
SMSS (B)	1448	1448	1448	1448	1448	1448	1448	1448	8948	8948	8948	8948
Stretch ACKs (bool)	false	false	false	true	false							
Flows (#)	1	2	8	8	16	16	16	16	16	16	16	16
Flow time (sec)	10	20	40	40	40	40	40	40	40	40	40	40
Flow pace (sec)	-	1	1	1	1	1	1	1	1	1	1	1
Target queue (us)	500	500	500	500	250	250	250	250	250	250	250	250
ECN (bool)	true	true	true	true	true	true	true	true	true	true	true	true
DCTCP g (weight)	1/16	1/16	1/16	1/16	1/16	1/16	1/16	1/16	1/16	1/16	1/16	1/16
Buffer capacity (ms)	200	200	200	200	200	200	200	200	200	200	200	200
RED Ramp Marking												
Min _{us}	500	500	500	500	250	250	250	250	250	250	250	250
Max _{us}	1500	1500	1500	1500	750	750	750	750	750	750	750	750
Max probability (%)	10	10	10	10	10	10	10	10	10	10	10	10
Burst (packets)	20	20	20	20	31	31	31	31	31	31	31	31
Min W _s	13.18	6.59	1.24	1.24	1.54	2.32	3.09	3.86	0.26	0.39	0.52	0.65
Max W _s	29.64	14.82	3.29	3.29	3.09	3.86	4.63	5.40	0.52	0.65	0.78	0.91
Adaptive	true	true	true	true	true	true	true	true	true	true	true	true
RED Step Marking												
K _{us}	500	500	500	500	250	250	250	250	250	250	250	250
W _s	13.18	6.59	1.24	1.24	1.54	2.32	3.09	3.86	0.26	0.39	0.52	0.65
RED Instantaneous Marking												
Min _{us}	500	500	500	500	250	250	250	250	250	250	250	250
Max _{us}	1000	1000	1000	1000	500	500	500	500	500	500	500	500
Min W _s	13.18	6.59	1.24	1.24	1.54	2.32	3.09	3.86	0.26	0.39	0.52	0.65
Max W _s	21.41	10.70	2.26	2.26	2.32	3.09	3.86	4.63	0.39	0.52	0.65	0.78

What to measure?

- Queueing delay, packet loss & link utilisation of the bottleneck³.
- Smoothed Round Trip Time (SRTT) of the sender.
- Packet marking rate of the AQM.
- Throughput of the individual sender.
- However, we do not address fairness.

³Uses techniques from here: <https://github.com/henrist/aqmt>

The framework

Which type of topology to use?

- Simple broadband testbed: two participating hosts; server and client topology.
- ECN initiating senders.
- Linux kernel stable version 5.0.

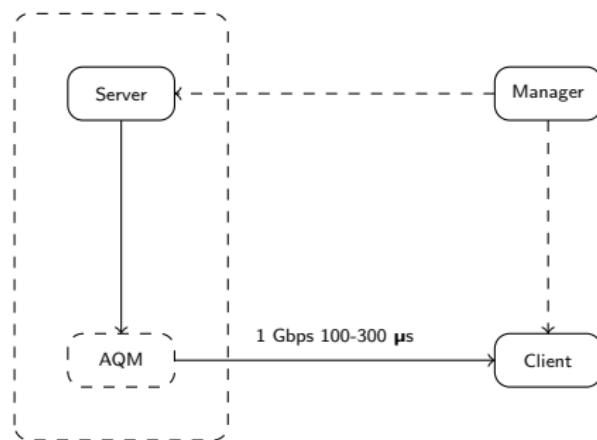


Figure: Experiment Setup

Design Proposal⁴

Requirements:

- Allow usage of lower congestion window values.
- Preserve acknowledgement clock.
- Avoid delayed acknowledgements at all cost.
- Scale down, rework, the additive increase during the congestion avoidance phase of TCP.

⁴The work given here is with considerable help from the supervisor. This design proposal continues the work of the submss paper [1], which first rediscovered and proposed a solution to the problem.

How to keep a congestion window of less than one segment?

Use fractional congestion window in addition to the normal window:

$$W_B = W_s * SMSS_B + W_{frac}$$

$$S_B = S_s * SMSS_B + S_{frac}$$

where W_{frac} and S_{frac} keeps a value $[0_B, SMSS_B]$.

Pros:

Cons:

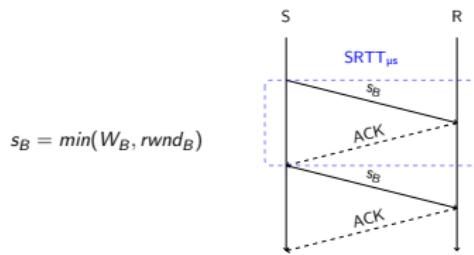
- Simple; No major modification required.
 - Use and update W_s and S_s as before (fully backward compatible).

- The Complexity to maintain a fraction.
 - W_B and S_B may require quad registers to know the full extend of the window.

How to keep fewer than one segment inflight per RTT?

Naive approach

- Use smaller packets when congestion window is less than one segment.
- The main idea behind packet slicing.



Cons:

- Conflicts with Nagle's algorithm.
- High cost for the network.
- New floor of 1 byte.
- Not easy to deploy beside classic TCP.

Pros:

- Simple concept.

How to keep fewer than one segment inflight per RTT?

Scalable approach

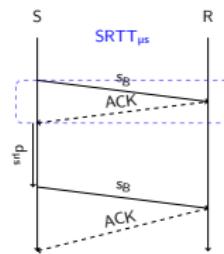
- Wait a local computed delay, $d_{\mu s}$, when congestion window is less than one segment.
- Send full packets if possible.
- Idea from the submss paper.

Pros:

- Overhead from network headers minimised.
- Lower computation cost per node.
- No new floor.

$$s_B = \min(SMSS_B, rwnd_B)$$

$$d_{\mu s} = \left(\frac{s_B}{W_B} - 1 \right) * SRTT_{\mu s}$$



Cons:

- Requires delay based transmission.

How to avoid delayed acknowledgements?

Naive approach

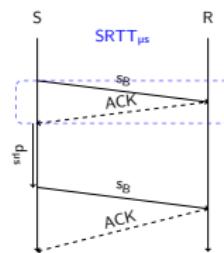
- Send segments in burst of the stretch acknowledgement factor, δ , to trigger an immediate acknowledgement.
- Idea based on work from Venkataramani et al.

Pros:

- The sender know the exact RTT to its receiver.
- Efficient loss recovery.
- TCP hardware offloading.

$$s_B = \min(\delta * SMSS_B, rwnd_B)$$

$$d_{\mu s} = \left(\frac{s_B}{W_B} - 1 \right) * SRTT_{\mu s}$$



Cons:

- AQM must maintain a deeper queue.
- Does not scale well for larger δ .

Stretch Acknowledgements

How to avoid delayed acknowledgements?

Scalable approach

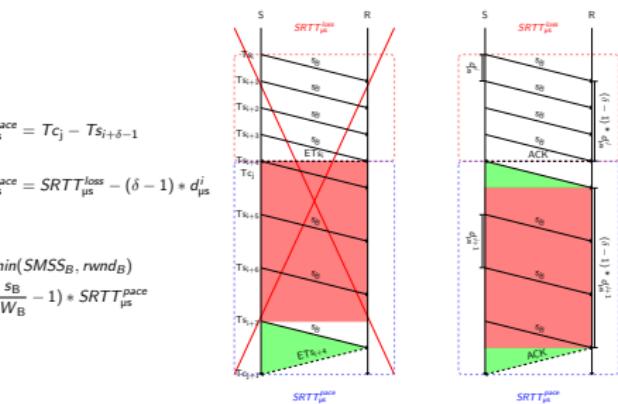
- Send sufficient segments paced.
- Subtract the accumulated delay afterwards.
- Idea based on work from Hsiao et al.

$$SRTT_{\mu s}^{pace} = Tc_j - Ts_{i+\delta-1}$$

$$SRTT_{\mu s}^{pace} = SRTT_{\mu s}^{loss} - (\delta - 1) * d_{\mu s}^i$$

$$s_B = \min(SMSS_B, rwnd_B)$$

$$d_{\mu s} = \left(\frac{s_B}{W_B} - 1\right) * SRTT_{\mu s}^{pace}$$



Pros:

- The sender know the exact RTT to its receiver.
- Loss resilient.

Cons:

- Not suited for hardware offloading.

Logarithmically Scaled Additive Increase Multiplicative Decrease (LS-AIMD)

How to probe for more capacity and perform congestion window reduction?

$$add_B \equiv C * k0 * lg(S_B/k1 + 1)$$

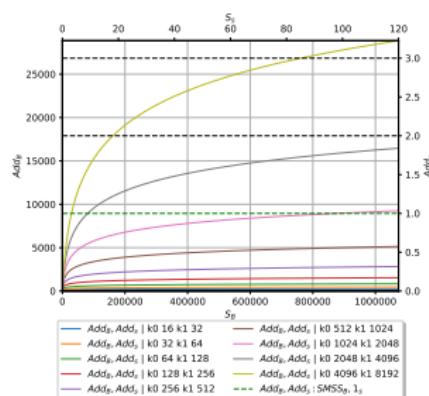
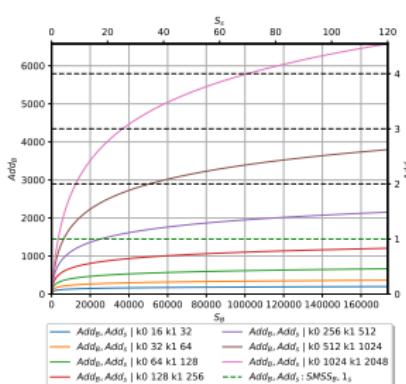
$$W_B \doteqdot add_B/W_s$$

$$W_{\mathrm{e}} \geq \delta$$

$$W_B \doteqdot add_B * SMSS_B / W_B.$$

$$W_{\varepsilon} < \delta$$

MTU_B	SMSS_B	k0	k1	K0	K1
1500	1448	256	512	8	9
9000	8948	1024	2048	10	11



Congestion window reduction

LS-AIMD Reno

$$S_B = \max(inflight_B/2, -2B)$$

LS-AIMD DCTCP

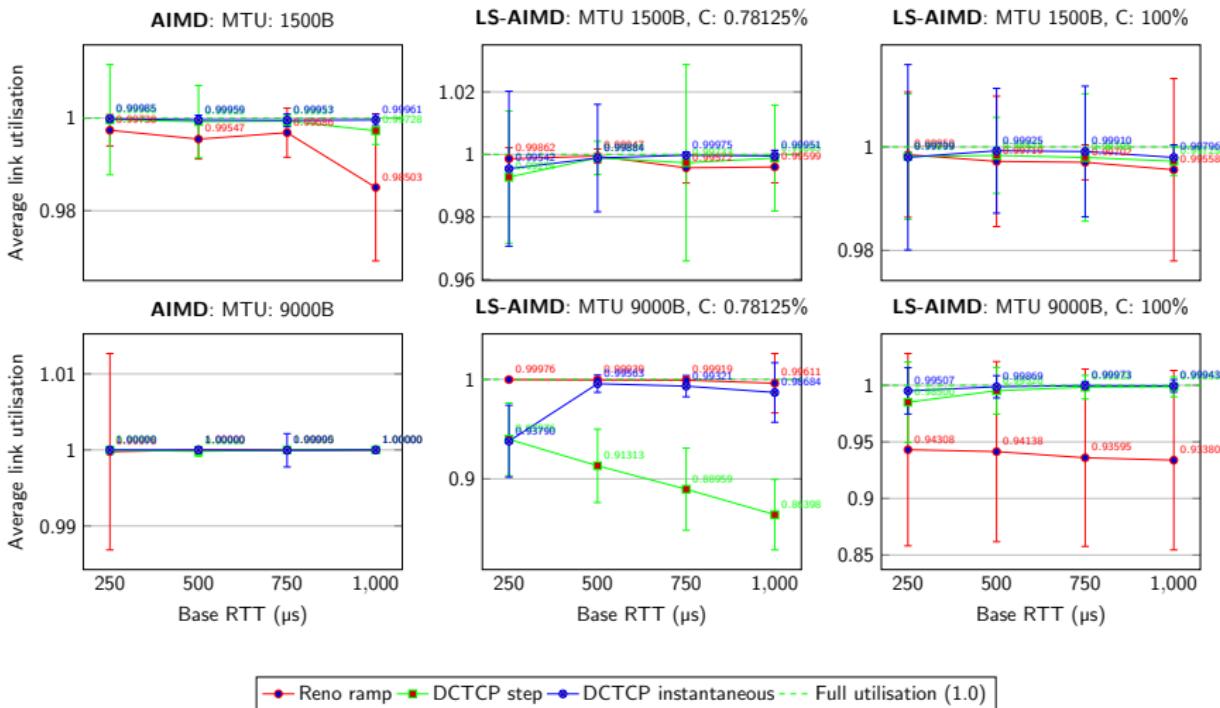
$$S_B = \max(inflight_B * (1 - \alpha/2), 2B)$$

Implementation

- Linux kernel v5.0⁵ (v4.13.16).
- Joint Linux kernel module for LS-AIMD Reno/DCTCP.
- Logarithmic function is currently rather complex.
- Scaled fractional congestion window.
- Packet conservation clock uses pacing to delay packet.
- Extends TCP to operate in two modes.
- Several optional hooks during packet processing.
- Stretch acknowledgement hardcoded: $\delta = 2$
- Not efficient at recovering loss.

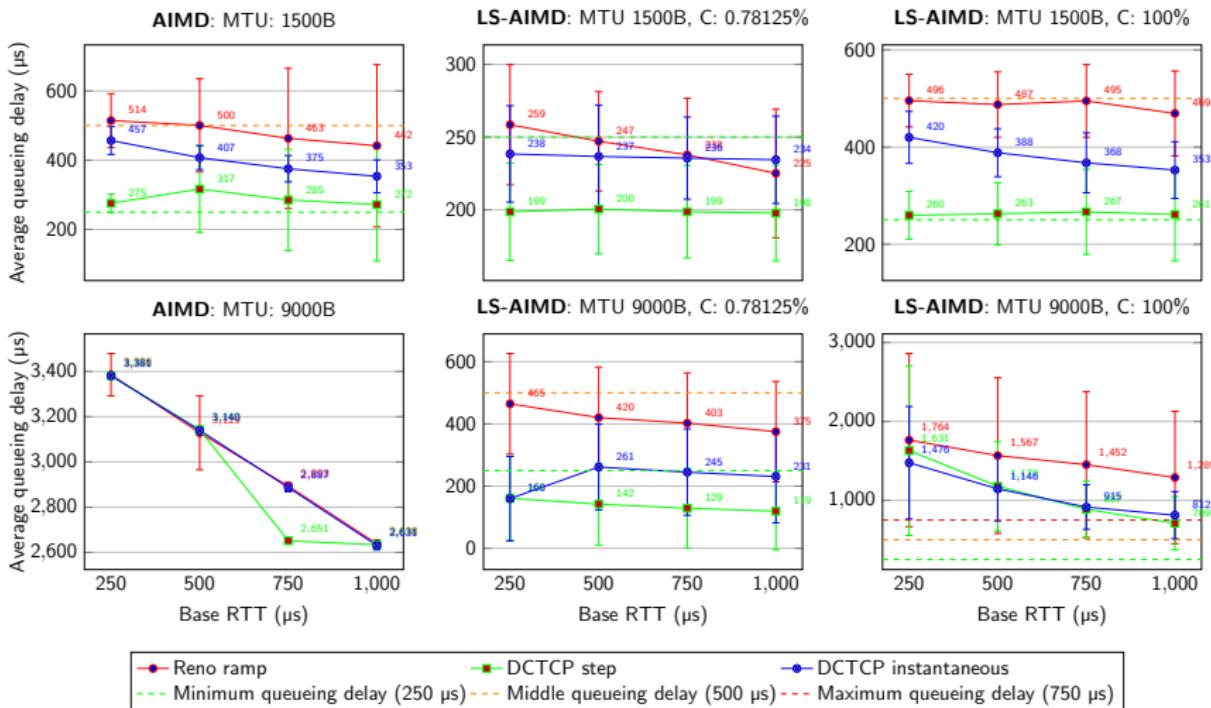
⁵Open source (GPL v2): <https://bitbucket.org/asadsa/kernel420/>

Did we keep the link utilised?



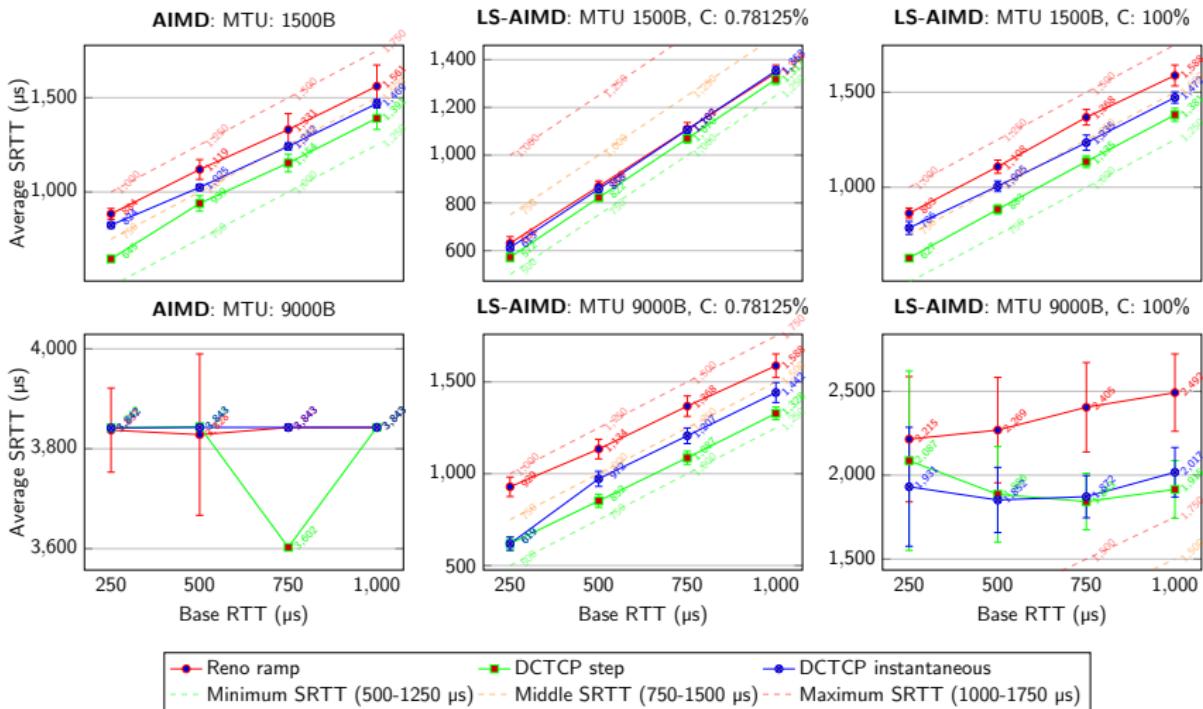
Queue Occupancy

If so, how much queue did the bottleneck have on average?



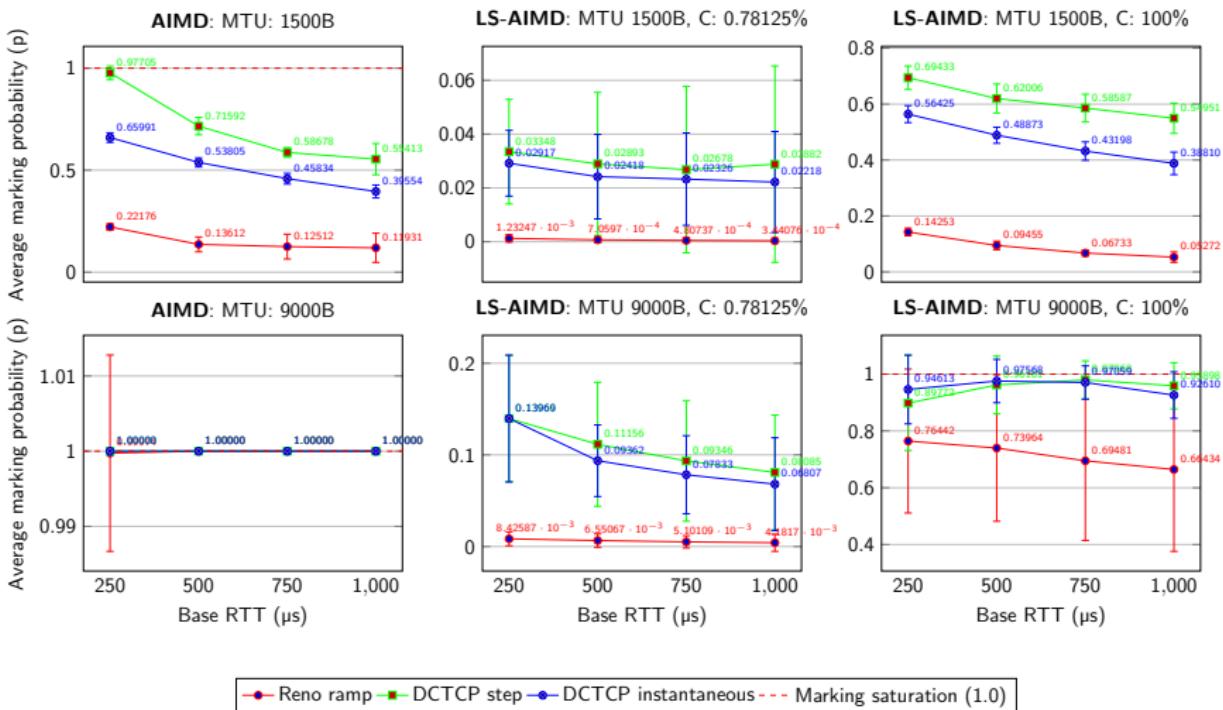
Smoothed RTT (SRTT)

What impact did the queueing delay have on the SRTT?



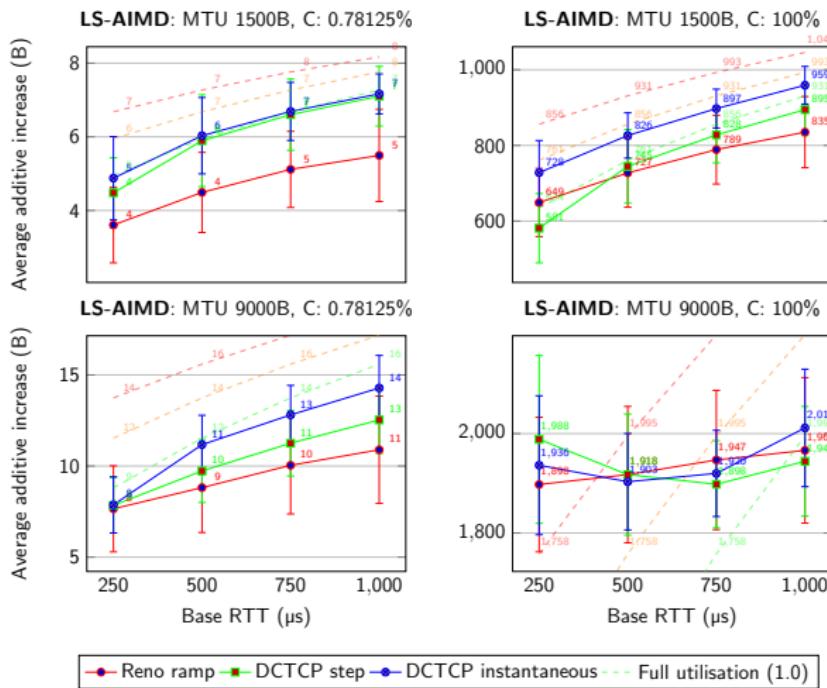
- Reno ramp
- DCTCP step
- ◆— DCTCP instantaneous
- - - Minimum SRTT (500-1250 μs)
- - - Middle SRTT (750-1500 μs)
- - - Maximum SRTT (1000-1750 μs)

Did the AQM tell about the ongoing congestion?



Additive Increase

Were we able to scale down the additive increase?



Throughput

How did the senders shareout the capacity?

MTU: 1500B, base RTT: 250μs

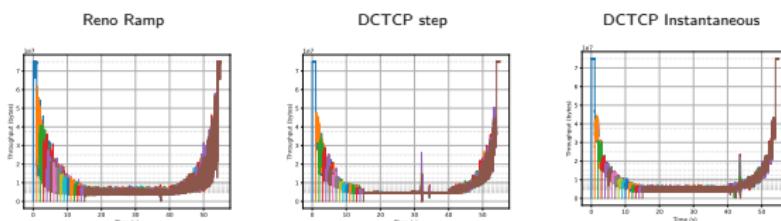
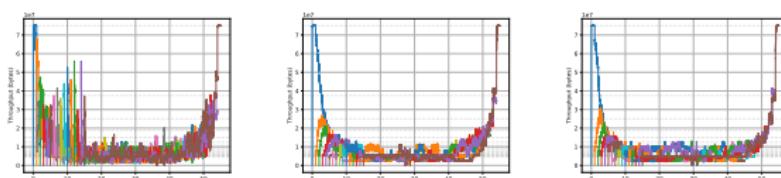
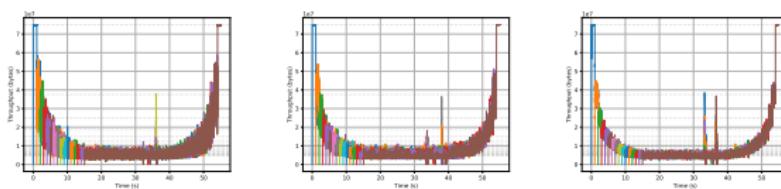
AIMD**LS-AIMD****C: 0.78125%****LS-AIMD****C: 100%**

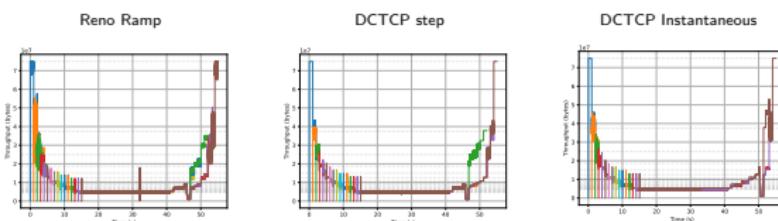
Figure: Experiment #4A: Throughput

Throughput

How did the senders shareout the capacity?

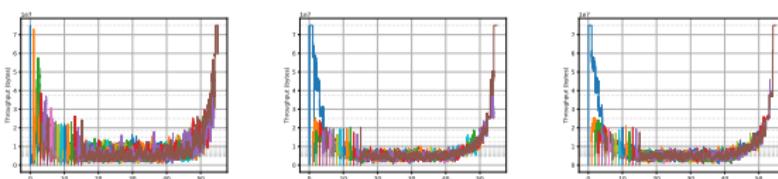
MTU: 9000B, base RTT: 250 μ s

AIMD



LS-AIMD

C: 0.78125%



LS-AIMD

C: 100%

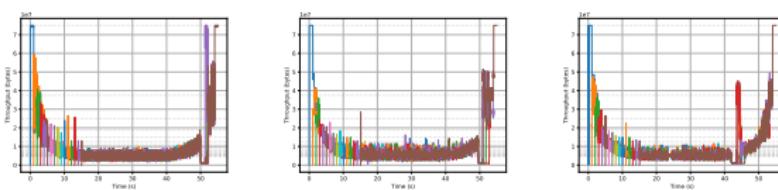


Figure: Experiment #4E: Throughput

Queueing Delay Trends

Who is best?

	Experiment (#)		1	2	3A	3B	4A	4B	4C	4D	4E	4F	4G	4H
RED Ramp Marking	AIMD Reno													
	Min (μs)	0	0	720	294	196	0	0	0	0	950	2162	2424	
	Max (μs)	1146	1572	6127	1998	917	1277	1474	1081	3768	3178	2916	2686	
	Mean (μs)	555	875	1067	1047	514	500	463	441	3385	3129	2893	2638	
	Standard deviation (μs)	212	214	145	188	77	134	202	234	93	163	15	16	
	95th percentile (μs)	917	1212	1310	1376	655	720	753	786	3407	3145	2916	2654	
	99th percentile (μs)	983	1343	1409	1507	688	786	819	851	3407	3178	2916	2654	
	LS-AIMD Reno (C: 0.78125%)													
	Min (μs)	-	-	-	-	0	98	0	0	0	0	0	0	0
	Max (μs)	-	-	-	-	688	1343	491	458	1114	1081	1114	1933	
RED Ramp Marking	Mean (μs)	-	-	-	-	258	247	237	225	464	420	402	375	
	Standard deviation (μs)	-	-	-	-	41	34	38	44	161	162	161	161	
	95th percentile (μs)	-	-	-	-	327	294	294	294	720	688	655	622	
	99th percentile (μs)	-	-	-	-	327	327	327	327	819	786	786	753	
	LS-AIMD Reno (C: 100%)													
	Min (μs)	0	131	393	393	0	0	0	0	0	0	0	0	0
	Max (μs)	1081	1572	1540	1605	1572	1769	1867	2818	3571	3178	2916	2686	
	Mean (μs)	571	866	1033	1016	495	487	495	469	1763	1567	1452	1289	
	Standard deviation (μs)	185	184	103	138	54	67	74	87	1096	987	923	839	
	95th percentile (μs)	851	1146	1212	1245	589	589	622	589	3342	3014	2654	2392	
	99th percentile (μs)	950	1277	1277	1343	622	622	655	655	3407	3145	2883	2555	

Queueing Delay Trends

Who is best?

Experiment (#)	1	2	3A	3B	4A	4B	4C	4D	4E	4F	4G	4H
RED Step Marking	AIMD DCTCP											
	Min (μs)	294	229	0	262	0	0	0	3145	2883	2162	2490
	Max (μs)	655	720	1048	1081	1048	4653	655	950	3473	3211	2686
	Mean (μs)	472	480	748	651	275	316	285	271	3381	3142	2651
	Standard deviation (μs)	50	83	14	155	26	125	146	162	15	13	10
	95th percentile (μs)	557	622	753	917	294	524	524	524	3407	3145	2654
	99th percentile (μs)	589	622	786	983	327	589	557	557	3407	3178	2654
	LS-AIMD DCTCP (C: 0.78125%)											
	Min (μs)	-	-	-	-	0	0	0	0	0	0	0
	Max (μs)	-	-	-	-	360	1703	753	720	917	950	884
	Mean (μs)	-	-	-	-	198	200	198	197	160	142	128
	Standard deviation (μs)	-	-	-	-	33	30	31	32	135	131	127
	95th percentile (μs)	-	-	-	-	262	229	229	229	425	393	393
	99th percentile (μs)	-	-	-	-	262	262	262	262	524	524	524
	LS-AIMD DCTCP (C: 100%)											
	Min (μs)	294	0	0	196	0	0	0	0	0	0	0
	Max (μs)	1605	655	7929	3309	1310	4325	1933	622	3407	3178	2916
	Mean (μs)	482	496	621	569	259	262	266	261	1630	1179	886
	Standard deviation (μs)	55	62	173	111	49	63	87	95	1075	565	357
	95th percentile (μs)	557	589	753	753	327	360	425	425	3375	2424	1441
	99th percentile (μs)	557	622	786	753	360	425	458	491	3407	3145	2359

Queueing Delay Trends

Who is best?

	Experiment (#)		1	2	3A	3B	4A	4B	4C	4D	4E	4F	4G	4H
RED Instantaneous Marking	AIMD DCTCP													
	Min (μs)	393	393	557	393	229	98	0	98	2785	2490	2392	2490	
	Max (μs)	655	786	1179	1310	622	819	819	753	3407	3178	4030	2686	
	Mean (μs)	546	584	885	893	456	407	375	353	3380	3139	2886	2630	
	Standard deviation (μs)	46	45	59	99	40	34	38	47	14	13	18	20	
	95th percentile (μs)	622	655	983	1048	524	458	425	425	3407	3145	2916	2654	
	99th percentile (μs)	622	688	1048	1114	557	491	458	491	3407	3145	2916	2654	
	LS-AIMD DCTCP (C: 0.78125%)													
	Min (μs)	-	-	-	-	0	0	0	0	0	0	0	0	0
	Max (μs)	-	-	-	-	491	5275	655	786	950	1114	1081	7012	
RED Instantaneous Marking	AIMD DCTCP													
	Mean (μs)	-	-	-	-	238	236	235	234	159	261	244	230	
	Standard deviation (μs)	-	-	-	-	33	35	28	30	135	137	138	149	
	95th percentile (μs)	-	-	-	-	294	294	262	262	425	491	491	458	
	99th percentile (μs)	-	-	-	-	294	294	294	294	557	589	589	589	
	LS-AIMD DCTCP (C: 100%)													
	Min (μs)	229	425	557	327	0	0	0	0	0	0	0	0	0
	Max (μs)	720	786	1114	2752	3211	1605	2490	1474	5210	3178	2916	2654	
	Mean (μs)	556	604	821	812	419	388	367	352	1476	1145	914	811	
	Standard deviation (μs)	37	45	58	101	53	49	61	58	713	406	281	299	
RED Instantaneous Marking	95th percentile (μs)	622	688	917	983	491	458	458	458	3211	1802	1343	1310	
	99th percentile (μs)	655	720	950	1015	524	491	491	491	3407	2850	1835	1998	

So what have we accomplished and where to next?

Main Contributions

- Made TCP perform well under shallow base RTT.
- Preserved stretch acknowledgements.
- A design proposal followed by a Linux kernel implementation.
- Extensive testing under very low base RTTs.

Future Work

- Conduct more experiments.
- Other network topologies.
- Look at other metrics.
- Autotune LS-AIMD for sub-packet regime.
- Measure the computational footprint of the code.

Further reading I



Bob Briscoe and Koen De Schepper. "Scaling tcp's congestion window for small round trip times". In: *Tech. rep., Technical report TR-TUB8-2015-002, BT* (2015).



Bob Briscoe, Koen Schepper, and Marcelo Bagnulo. *Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture*. Internet-Draft draft-ietf-tsvwg-l4s-arch-01.

<http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-l4s-arch-01.txt>. IETF Secretariat, Oct. 2017. URL:

<http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-l4s-arch-01.txt>.



Jay Chen. "Re-architecting Web and Mobile Information Access for Emerging Regions". AAI3482862. PhD thesis. New York, NY, USA, 2011. ISBN: 978-1-267-04868-4.

Further reading II



Jay Chen, Janardhan Iyengar, Lakshminarayanan Subramanian, and Bryan Ford. "TCP Behavior in Sub Packet Regimes". In: *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '11. San Jose, California, USA: ACM, 2011, pp. 157–158. ISBN: 978-1-4503-0814-4. DOI: [10.1145/1993744.1993804](https://doi.acm.org/10.1145/1993744.1993804). URL: <http://doi.acm.org/10.1145/1993744.1993804>.



Jay Chen, Lakshmi Subramanian, Janardhan Iyengar, and Bryan Ford. "TAQ: Enhancing Fairness and Performance Predictability in Small Packet Regimes". In: *Proceedings of the Ninth European Conference on Computer Systems*. EuroSys '14. Amsterdam, The Netherlands: ACM, 2014, 7:1–7:14. ISBN: 978-1-4503-2704-6. DOI: [10.1145/2592798.2592819](https://doi.acm.org/10.1145/2592798.2592819). URL: <http://doi.acm.org/10.1145/2592798.2592819>.

Further reading III



Inho Cho, Keon Jang, and Dongsu Han. "Credit-Scheduled Delay-Bounded Congestion Control for Datacenters". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '17. Los Angeles, CA, USA: ACM, 2017, pp. 239–252. ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098840. URL: <http://doi.acm.org/10.1145/3098822.3098840>.



W. -. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. "A self-configuring RED gateway". In: *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*. Vol. 3. Apr. 1999, 1320–1328 vol.3. DOI: 10.1109/INFCOM.1999.752150.



Wuchang Feng, Dilip Kandlur, Debanjan Saha, and Kang Shin. "Techniques for eliminating packet loss in congested TCP/IP networks". In: *U. Michigan CSE-TR-349-97* (1997).



Sally Floyd and Kevin Fall. "ECN Implementations in the NS Simulator". In: (1998).

Further reading IV



Pai-Hsiang Hsiao, H. T. Kung, and Koan-Sin Tan. "Video over TCP with Receiver-based Delay Control". In: *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV '01. Port Jefferson, New York, USA: ACM, 2001, pp. 199–208. ISBN: 1-58113-370-7. DOI: [10.1145/378344.378372](https://doi.acm.org/10.1145/378344.378372). URL: <http://doi.acm.org/10.1145/378344.378372>.



Pai-Hsiang Hsiao, HT Kung, and Koan-Sin Tan. "Streaming video over TCP with receiver-based delay control". In: *IEICE transactions on communications* 86.2 (2003), pp. 572–584.



J. Huang, Y. Huang, J. Wang, and T. He. "Adjusting Packet Size to Mitigate TCP Incast in Data Center Networks with COTS Switches". In: *IEEE Transactions on Cloud Computing* (2018), pp. 1–1. ISSN: 2168-7161. DOI: [10.1109/TCC.2018.2810870](https://doi.org/10.1109/TCC.2018.2810870).

Further reading V



J. Huang, Y. Huang, J. Wang, and T. He. "Packet Slicing for Highly Concurrent TCPs in Data Center Networks with COTS Switches". In: *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*. Nov. 2015, pp. 22–31. DOI: [10.1109/ICNP.2015.39](https://doi.org/10.1109/ICNP.2015.39).



I. Komnios, A. Sathiaseelan, and J. Crowcroft. "LEDBAT performance in sub-packet regimes". In: *2014 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. Apr. 2014, pp. 154–161. DOI: [10.1109/WONS.2014.6814738](https://doi.org/10.1109/WONS.2014.6814738).



H. T. Kung, Koan-Sin Tan, and Pai-Hsiang Hsiao. "TCP with sender-based delay control". In: *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*. July 2002, pp. 283–290. DOI: [10.1109/ISCC.2002.1021691](https://doi.org/10.1109/ISCC.2002.1021691).



M. Miao, P. Cheng, F. Ren, and R. Shu. "Slowing Little Quickens More: Improving DCTCP for Massive Concurrent Flows". In: *2015 44th International Conference on Parallel Processing*. Sept. 2015, pp. 689–698. DOI: [10.1109/ICPP.2015.78](https://doi.org/10.1109/ICPP.2015.78).

Further reading VI



R. Morris. "TCP behavior with many flows". In: *Proceedings 1997 International Conference on Network Protocols*. Oct. 1997, pp. 205–211. DOI: [10.1109/ICNP.1997.643715](https://doi.org/10.1109/ICNP.1997.643715).



Pai-Hsiang Hsiao, H. T. Kung, and Koan-Sin Tan. "Active delay control for TCP". In: *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*. Vol. 3. Nov. 2001, 1626–1631 vol.3. DOI: [10.1109/GLOCOM.2001.965855](https://doi.org/10.1109/GLOCOM.2001.965855).



Lili Qiu, Yin Zhang, and Srinivasan Keshav. "Understanding the Performance of Many TCP Flows". In: *Comput. Netw.* 37.3-4 (Nov. 2001), pp. 277–306. ISSN: 1389-1286. DOI: [10.1016/S1389-1286\(01\)00203-1](https://doi.org/10.1016/S1389-1286(01)00203-1). URL: [http://dx.doi.org/10.1016/S1389-1286\(01\)00203-1](http://dx.doi.org/10.1016/S1389-1286(01)00203-1).



Arun Venkataramani, Ravi Kokku, and Mike Dahlin. "TCP Nice: A Mechanism for Background Transfers". In: *SIGOPS Oper. Syst. Rev.* 36.SI (Dec. 2002), pp. 329–343. ISSN: 0163-5980. DOI: [10.1145/844128.844159](https://doi.org/10.1145/844128.844159). URL: <http://doi.acm.org/10.1145/844128.844159>.