# P4-enabled Network-assisted Congestion Feedback: A Case for NACKs

Anja Feldmann, Balakrishnan Chandrasekaran
Seifeddine Fathalli, Emilia N. Weyulu
Max-Planck-Institut für Informatik
{anja,balac,fathalli,eweyulu}@mpi-inf.mpg.de

## ABSTRACT

There exists an extensive body of work, spanning more than two decades, on congestion control schemes and signaling mechanisms. The majority of prior work does not, however, entertain the notion of network-assisted feedback for congestion control. The scope of the remaining work has also been, unfortunately, rather narrow: Some efforts limit themselves to using weak signals (involving a few bits in the header) and relying on receivers to reflect such signals to the sender; few others maintain per-flow statistics or explicitly set the rates the senders should use. Virtually all suggested network-assisted congestion feedback mechanisms are ineffective, not scalable, or limited to data-center contexts.

In this proposal, we exploit data-plane programmability of P4 switches as well as hardware-supported priority classes to present a novel network-assisted congestion feedback (NCF) mechanism. The feedback entails NACKs that are directly sent to the sender, and does not involve the receiver; it is, hence, quick and efficient. We propose sending such NACKs during periods of congestion to senders of elephant flows and outline a scalable approach to identify elephant flows. Unlike prior work, NCF is applicable to both data-centers as well as Internet-wide.

## CCS CONCEPTS

• **Networks** → **Packet classification**; **Programmable networks**; **In-network processing**;

## KEYWORDS

Congestion control, P4, AQM, NACKs

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*BS '19, December 2–3, 2019, Palo Alto, CA, USA*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7745-4/19/12…$15.00
https://doi.org/10.1145/3375235.3375238

## 1 INTRODUCTION

Congestion control is responsible for avoiding congestion collapse and is one of the most challenging tasks in the Internet [5, 25, 29]. Effectively controlling the congestion is becoming increasingly difficult, e.g., [3, 23], due to availability of very-high-speed links, increases in traffic diversity and burstiness, decreases in buffer sizes (relative to link speed), and the needs of the congestion-control (CC) mechanism to meet diverse goals. Today's goals focus not only on fair sharing of network resources, e.g., [35], but also on lowering delays, maximizing throughput, and effectively solving the Incast challenge (e.g., [3, 9, 11, 14]).

It is, therefore, not surprising that there is a large body of work on congestion control, including numerous CC schemes (e.g., Cubic [22], the default CC mechanism in Linux kernels, and BBR [9]), congestion-signaling mechanisms (e.g., ECN [28]) and many data-center-specific (e.g., DCTCP [3], pFabric [4], PCC [13, 14], QJUMP [20], NDP [23], Copa [6], and Homa [36]) as well as application-specific CC schemes (e.g., QUIC [30]). Huang et al. [24] presents an in-depth survey of this solution space. Recent approaches have also proposed the use of machine learning algorithms to dynamically tune the CC mechanism to achieve the optimal performance in a given scenario [31, 45].

The idea of eliciting support from the network to improve end-to-end CC schemes is not new (e.g., [2, 15, 27, 32, 37]). Scope of prior work in this space, however, has been rather narrow: Prior efforts either restrict themselves to using only a few bits for signaling (e.g., ECN [42], SNA [18], DECbit [43], and ATM [34]) or to setting explicit rates for senders (e.g., [37] and RCP [15]). While the former is an insufficient signal and also does not guarantee that the signal will affect only the source(s) responsible for congestion, the latter per-flow mechanism is simply not scalable. Even other approaches that accommodate rich congestion signals (e.g., [27, 32]) rely on receivers reflecting such signals back to the senders, implying a delayed congestion-feedback loop.

Among the reccurring takeaways of prior CC work are the following four observations: (1) We require mechanisms to handle both short flows and long flows, typically referred to

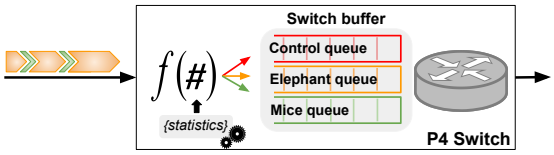



**Figure 1: *Overview of the network-assisted congestion feedback mechanism***

as *mice* and *elephants*, respectively [1, 21]; while mice dominate in terms of the number of flows, elephants dominate by volume of data transferred [17]. (2) Sharing queues between elephants and mice adversely affects the performance of flows and prioritization of traffic helps in alleviating the issue [20]; today, priority classes and multiple queues are supported even in commodity hardware. (3) Buffers should be kept short to ensure short, deterministic delays. (4) Buffers are getting shallow, i.e., their size relative to link speed is getting smaller [3, 7].

With advancements and adoption of programmable data planes [8], we propose a *network-assisted congestion feedback (NCF)* mechanism that exploits the above observations and offers an effective means to control congestion regardless of whether it is in a data-center or Internet-wide scenario. We exploit the programmability of today's switches to identify elephant flows independent of queue length and send explicit congestion notifications (NACKs) to the sender of these flows for throttling them. We are, in a limited form, resurrecting an idea similar to that of source quench [40] or choke messages [38], but *without* the caveats and issues that resulted in the deprecation of those old ideas (e.g., [19]). We allay security concerns, e.g., switch overload, by highlighting that the generation of the feedback signal (the NACKs) is done entirely *within* the data plane.

We summarize the contributions of this proposal as follows. We maintain three separate queues—two for mice and elephant flows, and one for control packets. We present a technique to identify, at line rate, without maintaining per-flow states, elephant flows using *rolling* sketches. We outline a simple work-conserving memory allocation scheme to manage the memory reserved for the three queues. Lastly, instead of dropping packets we use network-generated NACKs to throttle elephant flows. The design of NCF (refer Fig. 1), in our opinion, can ensure fair sharing of resources across both mice and elephant flows, requires only short queues, achieves high throughput, and tackles the Incast problem. The design of NCF, furthermore, lends itself to be applicable to both data-centers and the Internet.

## 2 DESIGN GOALS

Traffic flows in the Internet can be broadly categorized into two types: (a) *elephant* flows, which involve the transfer of a relatively large volume of data and last for a relatively longer duration; and (b) *mice* flows, which constitute the remaining flows of shorter durations and data-transfer sizes. While the users' expectation is simply minimal flow-completion times (FCTs), regardless of flow types, operators have to tackle at least two key challenges: (a) realizing the best performance from the existing network infrastructure; and (b) planning and provisioning capacity when, and where, required; over-provisioning hardware is not the panacea for all network problems. To meet both the users' and operators' expectations, CC solutions have to meet several criteria.

• ***Fair sharing of resources.*** A key objective is to facilitate fair sharing of bandwidth between all flows including both mice and elephant flows. In particular, elephant flows should not progress at the cost of mice flows.

• ***Short queues or buffers.*** Network queues or buffers allow a network element to handle transient traffic spikes or congestion. Moreover, provisioning large buffers will not solve these problems, and is typically undesirable, or infeasible, in practice. Large buffers, for instance, introduce additional delays; for mice flows most of their FCTs will constitute time wasted in such buffers. Provisioning large buffers, especially for high-speed links, can be prohibitively expensive: Per-port on-chip memory is rather expensive.

• ***Handling TCP Incast.*** Incast is the TCP performance degradation issue that arises when a large number of senders simultaneously send traffic to one receiver. Such a many-to-one traffic pattern naturally arises in storage and distributed systems (e.g., distributed file systems and map-reduce applications) [39]. Packet losses at the bottleneck link affect many senders, all of whom then wait for a random period (typically, one retransmission timeout of 200 ms) before attempting retransmissions. Thus, the problem adversely affects not just the FCTs of many senders, but also the completion time of the distributed job or computation.

• ***Work-conserving design.*** Statically assigning priorities or queue sizes for mice and elephant flows might not suit different kinds of workloads or traffic patterns. Thus, a work-conserving strategy is needed to maximize switch throughput.

• ***Scalability.*** Tracking the state of every flow quickly becomes infeasible in large networks [16], due to the required amount of memory. It is, hence, crucial to avoid or minimize tracking of flows to ensure scalability.

## 3 INSIGHTS

To meet the design goals outlined in the prior section we exploit the following insights.

• ***Address the elephants.*** Elephant flows contribute most to bytes transferred, albeit most flows in the Internet are mice. While elephant flows are subject to congestion control, mice flows (due to their size and duration) are *not*:

Mice flows may fillup a network buffer or queue at any time [26].

• ***Three queues suffice.*** It is well-known that sharing a queue between elephant and mice flows adversely impacts the FCTs of all flows [20]. Rather than statically assigning flow priorities, we estimate them online, in the data plane. We show that three queues suffice to assist end-to-end congestion control schemes. Note, that today's switches and routers offer hardware support for a small number of queues.

• ***Detect elephants in data plane.*** Data plane programmability facilitates the use of advanced data structures such as sketches (e.g.,[10, 12]) to compute key statistics [47]. Essentially, detection of heavy hitters (or elephant flows) is not feasible entirely in the data plane [44].

• ***Trim packets in data plane.*** The idea of trimming packets, by stripping the payload, in data-center environments as demonstrated by [23, 41] can be reused even in an Internet-wide context.

• ***Network (switches/routers) can initiate NACKs.*** Until now, network assistance, for end-to-end congestion control, has been restricted to the use of adding and updating one or two bits in a packet. Furthermore, since the network is not directly signaling the sender, the sender relies on the reflection of such signals by the receiver. P4 enables us to send network congestion feedback directly to the sender via NACKs, and, by removing the receiver from the feedback loop, this can drastically reduce the feedback duration (and sender's reaction time).

• ***Dynamically allocate memory.*** Rather than statically allocating memory for a given network queue, switches can dynamically manage the amount of memory allocated to different queues (following a work-conserving design).

• ***Congestion signals independent of queue usage.*** It is possible to send congestion signals to heavy elephants even if they do not currently have any packets queued at a switch.

## 4 THE DESIGN OF NCF

In this section, we discuss the implementation of NCF and highlight our key design choices.

### 4.1 Rolling Sketches

Ideally, to quickly mitigate congestion experienced at a switch, we must send congestion notifications to a subset of the elephant flows inbound to the switch; recall, from §3, that elephant flows are subject to congestion control, whereas mice are not. Sending such notifications only to the sources (or senders) of elephant flows necessitates the identification of elephant flows, continuously, at any point in time. To this end, we exploit the novel capabilities of P4 switches, namely the capability to maintain counters and various kinds of sketches at line rates [33, 44]. Specifically, we build and continuously update, at line rate, a *rolling* (count) sketch that facilitates dynamic identification of elephants without keeping state for all flows.

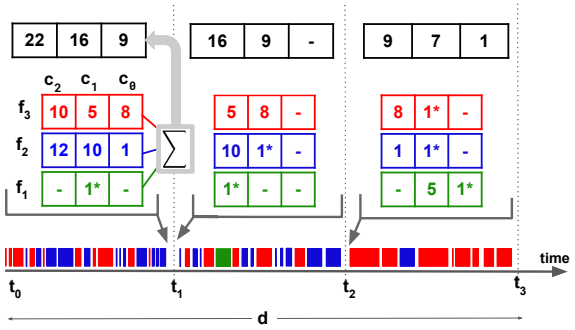



**Figure 2: *Illustration of rolling counters***

The idea of rolling sketches is similar to that of rolling counters. Sketches, in contrast to counters, provide good estimates for the high-frequency items (in this case, the elephant flows) while using very limited memory. Given our objective to detect and track elephants, sketches are ideal for our use case. The rolling sketches can also handle the transition of mice flows into elephant flows, and vice versa.

*Tracking flows.* We maintain a counter $c^t_{f_i}$ per flow and time window. Rather than tracking all time windows, however, we maintain $n$ windows with each representing a duration $d$. These $n$ windows then allow us to track flow popularity over a time period of up to $d \times n$. At time $t_0$, the $0^{\text{th}}$ time window is used to count the packets. The flow counters $c^0_{f_i}$ capture, hence, the packets arriving within the time window $[\lfloor t/d \rfloor * d, (\lfloor t/d \rfloor + 1) * d]$. More generally, the flow counters $c^j_{f_i}$ capture packets arriving within the window $[(\lfloor t/d \rfloor - j) * d, (\lfloor t/d \rfloor - j + 1) * d]$. In addition to these counters, we track the total traffic within each time window $j$: $T_j = \sum_{f_i} c^j_{f_i}$.

*Detecting elephants.* To check whether a flow $f_i$ is an elephant, we define a function $F$ that takes as input the per-flow counters $c^j_{f_i}$ and the total-traffic estimates $T_j$ and returns a boolean. An average threshold-based (say, 5%) function, for instance, is one simple example to realize $F$. In this example, if a flow contributes more than 5% on average[1] within the last $n$ periods of duration $d$, we label it an elephant; otherwise, we label it mice. If time $t$ crosses an interval boundary, the current flow counters are moved from index 0 to 1, and the counting resumes on a set of newly initialized counters.

*Visualizing rolling counters.* From left to right, the illustration, in Fig. 2, shows the states of three counters $\{c_2, c_1, c_0\}$

[1]Another implementation of $F$ could require an elephant to exceed the threshold only in one or in all time intervals. There are, hence, many ways to define $F$, and it can be customized based on our objectives.

at different points of time. Right before the interval boundary at $t_1$, flows $f_2$ and $f_3$ contributed 1 and 8 packets (in the column corresponding to $c_0$), respectively. The flow $f_2$ ($f_3$) contributed 5 (10) packets in the prior interval, and 10 (12) packets in the oldest interval, which is two intervals before the current interval. Both these flows would be labeled elephants. Flow $f_1$, in contrast, contributed nothing except for the prior interval where it contributed 1 packet. Therefore, it would be labeled a mice. Per this illustration, note that after crossing the time interval $t_1$, the counter values are "rolled" over to the left; values in $c_0$ are moved to $c_1$, that in $c_1$ to $c_2$, and $c_0$ is re-initialized. The counter values at $t_2$ indicate that $f_2$ has transitioned from being an elephant to being a mice, while $f_3$ has transitioned in the opposite manner.

*From counters to sketches.* Per-flow counters are not scalable due to the amount of memory they require. We can, however, use sketches [10, 12] to approximate the counts. We, therefore, replace the counters for each time period with a sketch (e.g., a CountMin sketch) that provides approximate counts, while requiring a small and fixed memory budget (regardless of the number of flows tracked). The switch from counters to sketches does not affect our ability to detect as well as discriminate elephant flows from mice flows.

## 4.2 Queues

Since the relative size of buffers (i.e., size of the buffer relative to link speed) in switches have been drastically decreasing [3, 7], we designed our CC mechanism to require only three short queues. As a side-effect, flows experience only low and deterministic delays.

We propose to use three queues: one for elephant flows, one for mice, and one for control packets. To classify and schedule incoming packets onto the queues, we do not require per-queue statistics. We rely, instead, on the sketches from the prior section (§4.1). We suggest the following "rule-of-thumb" reserved sizes for the three queues to ensure a minimum bandwidth for the different packets. Allocate 10% each to mice and elephant queues, and a maximum of 5% to the control packet queue.[2] The remaining bandwidth is distributed in a work-conserving manner as needed. Our bandwidth reservations together with using a work-conserving scheduler ensures that elephant flows cannot starve mice flows and vice versa.

We suggest redistributing memory on demand and use the reservations only when available memory becomes a bottleneck. Our goal is to allow mice and/or elephant flows to use up almost all available memory (up to a limit, e.g., 95%), and, consequently, the bandwidth, if there are no other packets in the switch.

[2]Multiple priority queues with bandwidth reservations are commonplace on network switch hardware today.

## 4.3 Network-assisted Congestion Feedback

When congestion occurs at a switch, marked by the lack of available memory for the queues, we send an explicit congestion signal or feedback (rather than marking packets to indicate congestion—à la ECN) to the source or sender. While these notifications are based on the idea of packet trimming from NDP [23], our design significantly diverges from that of NDP: We generate a NACK from the trimmed packet and send it directly to the sender. Both packet trimming and NACK generation can be performed efficiently within the data plane, using P4, as it only entails switching source and destination endpoints (i.e., IP address and ports) and adding a flag to mark the packet as a NACK.

By sending the NACK directly to the sender, we remove the receiver from the feedback loop. Furthermore, we queue the NACKs in the highest-priority control queue. Our congestion notifications, as a result, can be expected to be even faster than NDP and ECN and, thus, can significantly reduce the sender's reaction time. The NACKs should also help tackle the Incast problem. A sender should be able to quickly reduce its sending rate and, thus, ensure a fair sharing of the network bandwidth. Lastly, NACKs also help the sender directly infer which packet did not reach the receiver (as a result of the trimming at the switch).

Although we do not combine multiple NACKs to the same source, the overhead in sending multiple NACKs is likely to be limited. Moreover, NACKs can be dropped if the network is overly congested.

## 4.4 Utilizing NCF

The utilization of NCF simply depends on how the switch memory is allocated between mice and elephant flows; memory required for buffering NACKs is minimal since NACKs only contain packet headers. Incast traffic, which constitutes mainly mice flows, requires tilting the memory allocation in favor of the mice queue. High-throughput elephant flows, in contrast, require the opposite—more memory for the elephant queue.

If support for Incast dominates that for high throughput flows (a necessity in many data-center scenarios), load spikes in mice packets should be accommodated. We may have to allocate, under such circumstances, more memory to the mice queue by reclaiming buffer from the elephant queue. Reclaiming memory entails dropping packets from the concerned queue. In this case, we also generate NACKs and send them to the sender via the control queue.

An obvious disadvantage of the above approach is that we only throttle elephant flows that currently have packets in the elephant queue. This strategy might not always result in throttling the most appropriate flows. Alternatively, we

can supplement our rolling sketches (refer §4.1) with a priority queue to retain information about the top $N$ (say, 20) elephant flows within the last $n \times d$ time period. In case of congestion, we can then send NACKs for these top elephant flows, regardless of whether they currently have packets in the queue, until the load across the switch decreases.

To avoid attacks, where an attacker might use a lot of mice flows to cause packet drops across elephant flows, we limit such throttling to a short time period. Note, we know how many mice packets have been handled within the last $n \times d$ time period, which should exceed a multiple of the expected RTT. If we observe that mice flows are using too many resources, we can again re-allocate the memory to maintain a balance between all flows.

## 5 DISCUSSION

NCF does not obviate the need for CC schemes; rather it calls for the control loop to simply acknowledge explicit signal from the network concerning congestion. Unlike prior work, NCF targets only elephant flows for such signals to simplify the design while maximizing effectiveness as well as scalability. We briefly outline a few key design choices and tradeoffs that require a thorough examination in the future.

*Tuning control knobs.* The presentation of NCF design (§4) includes various recommendations or "thumb-rules" on the choice of thresholds, queue sizes, and data structures. These recommendations can be refined empirically via extensive experiments against a host of CC schemes (e.g., via Pantheon [46]) or adjusted based on the needs and expectations of a network operator, or even dynamically altered using off-the-shelf machine-learning tools.

*Interplay of control loops.* There is a rich body of work, spanning decades, on congestion-control schemes which include a wide spectrum of optimizations for detecting, mitigating, and avoiding congestion on the path between the sender and receiver. Even though NCF technically does not introduce an additional control loop, the sending of NACKs to the sender shortens the control loop and its impact on existing CC schemes at the endpoints require a thorough evaluation. However, we expect that congestion control schemes at the endpoints can be simplified or made more efficient by taking into account the explicit network information via the NACKs.

*Security considerations.* We acknowledge that the ability to control a sender's rate via NACKs raises a few security concerns, similar to that of "source quench" [19]. We note, however, that the likelihood of a flow getting affected depends significantly upon its duration. Only the small fraction of elephant flows are likely affected. Moreover, an attack requires the attacker to guess the five-tuple identifier as well as the sequence number. The attack is, hence, no different from generating fake or duplicate ACKs—NCF does not per-se introduce any new problems. Since the NACKs are generated at line rate within the data plane, we do not introduce any new opportunity (vulnerability) to overload the switch.

*Congestion visibility.* The visibility of NACKs to any third-party, who can observe the traffic, might be immensely helpful in both research and practice. Since the feedback (i.e., NACKs) are generated in the data plane, at the switch where congestion was experienced, there is no fundamental limitation restricting us from augmenting the feedback signal with additional information (e.g., the switch identifier or IP address). This supplementary information, however, may leak information about which networks are experiencing congestion, and also where; operators do not have strong incentives for providing such supplementary data.

## 6 CONCLUSION

A host of techniques have been explored until now to provide an effective solution for congestion control. Virtually all prior work have either never considered including network assistance in the end-to-end congestion control process or limited the use of such signals to one or two bits (e.g., ECN) or their applicability to data-center or layer-2 environments.

We leveraged hardware support in switches for a few queues and the increasing adoption of P4 to present a network-assisted congestion feedback mechanism. Our design uses only three queues in a P4 switch and proposes NACKs as an explicit network congestion feedback. We exploit the capabilities of a P4 switch to dynamically track elephant flows and throttle them, as required, using NACKs. The NACKs, sent directly to the sender, are expected to arrive at the sender faster than prior work (e.g., ECN and NDP) and should significantly reduce the sender's reaction time. Lastly, NCF is applicable in both data-center and Internet-wide contexts. We intend to follow up this proposal with prototype implementations on P4 switches and evaluate the various design choices and parameters.

## REFERENCES

[1] Ahmed M Abdelmoniem and Brahim Bensaou. 2015. Reconciling mice and elephants in data center networks. In *IEEE Int. Conf. on Cloud Networking (CloudNet)*. IEEE.

[2] Mohammad Alizadeh, Berk Atikoglu, Abdul Kabbani, Ashvin Lakshmikantha, Rong Pan, Balaji Prabhakar, and Mick Seaman. 2008. Data center transport mechanisms: Congestion control theory and IEEE standardization. In *Annual Allerton Conference on Communication, Control, and Computing*.

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *ACM SIGCOMM*.

[4] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pfabric: Minimal near-optimal datacenter transport. In *ACM SIGCOMM Computer Communications Review (CCR)*, Vol. 43.

[5] M. Allman, V. Paxson, and W. Stevens. 1999. TCP Congestion Control. RFC 2581. (April 1999).

[6] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[7] Wei Bai, Kai Chen, Shuihai Hu, Kun Tan, and Yongqiang Xiong. 2017. Congestion control for high-speed extremely shallow-buffered datacenter networks. In *Asia-Pacific Workshop on Networking*. ACM, 29–35.

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communications Review (CCR)* 44, 3 (2014).

[9] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control. (2016).

[10] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding Frequent Items in Data Streams. In *Int. Col. on Automata, Languages and Programming (ICALP)*.

[11] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. 2009. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *ACM Workshop on Research on Enterprise Networking (WREN)*.

[12] Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *J. Algorithms* 55, 1 (April 2005).

[13] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. 2015. {PCC}: Re-architecting Congestion Control for Consistent High Performance. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[14] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. {PCC} Vivace: Online-Learning Congestion Control. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[15] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown. 2005. Processor Sharing Flows in the Internet. In *Proceedings of the 13th International Conference on Quality of Service (IWQoS'05)*.

[16] Cristian Estan and George Varghese. 2003. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. *ACM Trans. Comput. Syst.* 21, 3 (Aug. 2003).

[17] Anja Feldmann, Jennifer Rexford, and Ramon Caceres. 1998. Efficient policies for carrying Web traffic over flow-switched networks. *IEEE/ACM Transaction on Networking (ToN)* 6, 6 (1998).

[18] Frederick D. George and Gerald E. Young. 1982. SNA flow control: Architecture and implementation. *IBM Systems Journal* 21, 2 (1982).

[19] Fernando Gont. 2012. Deprecation of ICMP Source Quench Messages. RFC 6633. (May 2012).

[20] Matthew P Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert NM Watson, Andrew W Moore, Steven Hand, and Jon Crowcroft. 2015. Queues Don't Matter When You Can {JUMP} Them!. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[21] Liang Guo and Ibrahim Matta. 2001. The war between mice and elephants. In *Int. Conf. on Network Protocols (ICNP)*. IEEE.

[22] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008).

[23] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *ACM SIGCOMM*.

[24] Shan Huang, Dezun Dong, and Wei Bai. 2018. Congestion control in high-speed lossless data center networks: A survey. *Future Generation Computer Systems* 89 (2018).

[25] Van Jacobson. 1988. Congestion avoidance and control. In *ACM SIGCOMM Computer Communications Review (CCR)*, Vol. 18.

[26] Youngmi Joo, Vinay Ribeiro, Anja Feldmann, Anna C. Gilbert, and Walter Willinger. 2001. TCP/IP Traffic Dynamics and Network Performance: A Lesson in Workload Modeling, Flow Control, and Trace-driven Simulations. *ACM SIGCOMM Computer Communications Review (CCR)* 31, 2 (April 2001).

[27] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion Control for High Bandwidth-delay Product Networks. In *ACM SIGCOMM (SIGCOMM '02)*.

[28] Srisankar Kunniyur and Rayadurgam Srikant. 2003. End-to-end congestion control schemes: Utility functions, random losses and ECN marks. *IEEE/ACM Transaction on Networking (ToN)* 11, 5 (2003).

[29] James F Kurose and Keith Ross. 2016. *Computer Networking: A top-down approach featuring the Internet, 7/E.* Pearson Education.

[30] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The quic transport protocol: Design and internet-scale deployment. In *ACM SIGCOMM*.

[31] Wei Li, Fan Zhou, Kaushik Roy Chowdhury, and Waleed M Meleis. 2018. QTCP: Adaptive congestion control with reinforcement learning. *IEEE Transactions on Network Science and Engineering* (2018).

[32] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *ACM SIGCOMM (SIGCOMM '19)*.

[33] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *ACM SIGCOMM*.

[34] Steven E Minzer. 1989. Broadband ISDN and asynchronous transfer mode (ATM). *IEEE Communications Magazine* 27, 9 (1989).

[35] Jeonghoon Mo and Jean Walrand. 2000. Fair end-to-end window-based congestion control. *IEEE/ACM Transaction on Networking (ToN)* 5 (2000).

[36] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A receiver-driven low-latency transport protocol using network priorities. In *ACM SIGCOMM*.

[37] Hiroyuki Ohsaki, Masayuki Murata, Hiroshi Suzuki, Chinatsu Ikeda, and Hideo Miyahara. 1995. Rate-based Congestion Control for ATM Networks. *ACM SIGCOMM Computer Communications Review (CCR)* 25, 2 (April 1995).

[38] R Pan and B Prabhakar. 1999. CHOKe: A stateless mechanism for providing Quality of Service in the Internet. In *Annual Allerton Conference on Communication, Control, and Computing*, Vol. 37.

[39] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. 2008. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In *USENIX Conf. on File and Storage Technologies (FAST)*. Berkeley, CA, USA.

[40] John Postel. 1981. Internet Control Message Protocol; RFC792. *ARPANET Working Group Requests for Comments* 792 (1981).

[41] Costin Raiciu and Gianni Antichi. 2019. NDP: Rethinking Datacenter Networks and Stacks Two Years After. *ACM SIGCOMM Computer Communications Review (CCR)* (Oct. 2019).

[42] K. Ramakrishnan, S. Floyd, and D. Black. 2001. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168. (September 2001).

[43] KK Ramakrishnan and Raj Jain. 1988. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. In *ACM SIGCOMM Computer Communications Review*

*(CCR)*, Vol. 18.

[44] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *ACM SIGCOMM Software Defined Networking Research (SOSR)*.

[45] Keith Winstein and Hari Balakrishnan. 2013. Tcp ex machina: Computer-generated congestion control. In *ACM SIGCOMM*.

[46] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[47] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic Sketch: Adaptive and Fast Network-wide Measurements. In *ACM SIGCOMM*.