The Origins of Congestion and Network Assisted End-to-End Congestion Control

Til Mohr
RWTH Aachen University
til.mohr@rwth-aachen.de

***Abstract*—In this article we present the most common origins of network congestion. To tackle congestion, different types of algorithms will be compared, by first describing how each perceives congestion and later on how they adjust to those circumstances. This paper will also take a closer look at network assisted congestion control and a real-use implementation in data centers.**

I. Introduction

Congestion was, is, and always will be a problem on the Internet, appearing in various forms with different symptoms causing minor, but also major challenges. Most data sent over the internet is split up in smaller packets that all try to reach their destination the fastest, whether it is by going the shortest or the least congested route. In an optimal network all computers, servers, etc. would have a direct link to each other, thus reducing the importance of bandwidth and consequently removing the need for buffers at all. However, the larger the network grows, the number of links would increase quadratically.

Networks today consist of network nodes (routers, switches, ...) which all try to route internet packets as quickly as possible whilst keeping the required hardware to a minimum. Ideally there would be a steady flow of packets being transmitted at maximum rate. However, when a internet node receives a packet while the outgoing connection still is in use, the packet has to be stored in a buffer until the outgoing connection freed up again. Therefore, those storage units are essential to every node on a network, but without proper management even well-sized buffers could lead to network problems like packet-loss, jitter, and connection delay [1].

This paper will present origins of network congestion and analyze how different types of algorithms try to avoid and minimize congestion.

II. Origins of Congestion

With the ever-growing amount of internet services not only data-centers, but the whole internet in general faces new challenges with each packet sent. Therefore, a network-wide reduction of those complications is an important task to improve the quality of all internet services.

Network delay can generally speaking be categorized into three different types of delay [2]:

- *Transmission delay* describes the delay of transmitting data packets between two internet nodes. Typically, packets sent from an origin to a destination node are sent over various different internet nodes, all of which could have a different bandwidth. Therefore, a packet send from **node 1** to **node n** over **nodes 2 to n-1** cannot be sent at a faster rate than the lowest bandwidth on the path of the packet (bottleneck) [Figure 1].

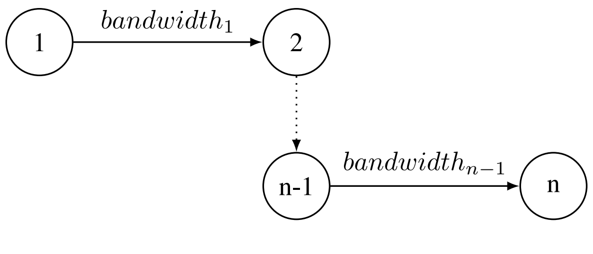

Fig. 1. Path a packet takes through the network. Each circle represents a network node with 1 being the sender and n being the receiver.

$$Rate_{max} \leftarrow \min\{bandwidth_i \mid i \in \{1, ..., n-1\}\}$$

- *Processing delay* is the time needed to process an individual packet by an internet node. This usually means processing the packets IP header, finding its destination and possibly queuing it in buffers.
- *Queuing delay* is the time a package spends waiting to be processed or transmitted.

Whereas processing delay is almost not noticeable and transmission delay could be improved by upgrading cables or transmission protocols, queuing delay has various causes and therefore is the most difficult one to reduce. There are many different ways to implement buffers in a network node and the effectiveness of each implementation varies from where it is used. Traffic in residential network nodes for example differs from nodes used in large data-centers in every aspect, thus hardware and software in those components should be adjusted accordingly.

Sizing buffers is a complex task. On the one hand read and write times should be as short as possible to minimize additional delay. On the other hand buffers should not be too small, as networks need to be robust for future changes and therefore nodes should store lots of data if needed, while also keeping in mind the requirements of congestion detection algorithms as described in section Congestion Avoidance in order to adjust for congestion appropriately [3]. Unifying those goals is very challenging as it leads to increasing costs and physical space occupied by those buffers. Additionally, a 2004

paper by Appenzeller et al. [1] found that 99% of buffers in network nodes, which were sized on a rule-of-thumb rule based on the TCP protocol [4], could be removed without losing, but rather gaining performance. Clearly, balancing speed and size is difficult and thus buffer sizes should be adjusted to their area of use accordingly.
This isn't the only major cause of queuing delay. Without proper buffer management even nodes with well-sized buffers will experience performance loss. Proper queue management is able to compensate for the unpredictable fluidity of network traffic. It is a difficult task to buffer the right amount of packages to keep transmission flow consistent over time. As the number of packets being transmitted grows, the *throughput* of the network node increases until packets are being transmitted at bottleneck rate. After this, the transmission rate cannot increase anymore, which leads to packets being buffered along their path to prevent them from being lost [2]. If the incoming amount of traffic keeps to exceed the amount of outgoing traffic, those buffers will fill up (*over-buffering*), which will eventually lead to *bufferbloat* [5], [6].

Bufferbloat

Bufferbloat refers to frequently full and excessively large buffers in internet nodes. This leads to partly unnecessary network delay and damages congestion avoidance algorithms, such as those found in classic TCP [2], [7]. Buffering is needed to provide space to queued packets waiting for transmission resulting in a minimization of packet loss. In the past however, buffer sizes were smaller than today's thus less packets could be queued and more were dropped, signaling transmission protocols the presence of congestion in this node early in. With memory getting cheaper over the years, buffers saw an increase in size. This has the negative impact, that more packets can be queued before any are dropped, thus those adjustments of congestion avoidance protocols kick in later than usual. Therefore, buffers are being filled in the whole network which results in an extreme increase in network delay (*latency*) and fluctuation (*jitter*) [2], [3], [7].
So Services, which require low latency at a high data-rate, such as streaming, will experience a decrease in quality.

III. Congestion Avoidance

As already mentioned, congestion detection algorithms make up an important part in congestion avoidance protocols. *TCP* is such a protocol, based on the transport layer, which adapts its send rate according to available network bandwidth [8], [9]. To fully understand, how TCP's congestion avoidance algorithm operates, one needs to understand how TCP works. TCP lives on the transport layer. It communicates with the application layer via sockets, splits up data into packets (and vice versa), and transmits those packages over the network with the help of the internet protocol, thus TCP is often also referred to as *TCP/IP*. It is the interface between network components and computer programs. Each of those packets have different purposes, but underlay the same structure - the protocol header. The TCP header is split up into multiple segments [Figure 2]:

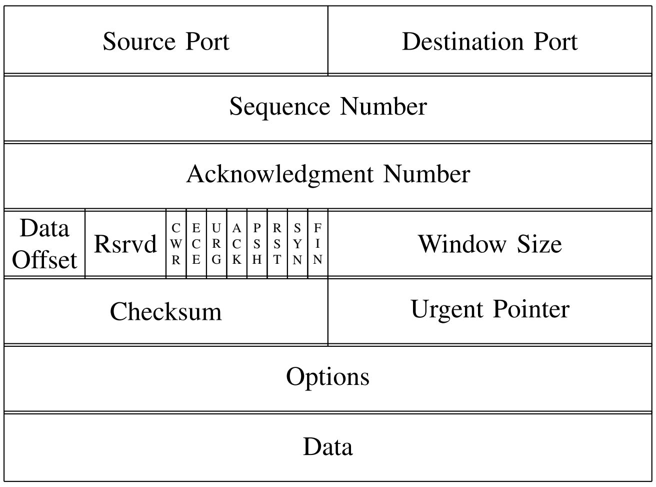

Fig. 2. Basic structure of the TCP header. It is split up into multiple 32-bit blocks. The data block contains all the application data to be transmitted [9], [10], [11].

- *Source Port* and *Destination Port* specify the ports on which TCP should communicate with the application layer.
- The *Sequence Numbers* and *Acknowledgment Numbers* specify how much data has been sent so far. All bytes in a TCP connection are numbered in increasing order, with both sides of the connection keeping track of those numbers. This way packet loss can be detected. As the sequence numbers are limited to 32 bit, they cannot increase endlessly. Thus, if all possible sequence numbers are used up, it wraps around to 0 to keep the connection up and running [12].
- TCP can set eight different bit-flags in its header. The most important one is the *ACK* flag (Acknowledgement bit). Whenever a data packet arrives at its destination, a packet acknowledging the arrival of this packet will be send back. This acknowledgement packet always has the *ACK* flag set. The *SYN* and *FIN* flags initialize and terminate a connection.
- The *Window Size* is the maximum amount of data (in bytes) a receiver is willing to receive at any time. It is important to set it appropriately to prevent unnecessary packet loss or delay, as the receiver couldn't process the data in time. Thus, it is will also be constantly adjusted to improve the connections reliability [9].

TCP uses a handshake system. Whenever a client sends a data packet to the server, the server waits until it has received enough data (specified by the window size), and then sends back an acknowledgment package, signaling the successful transmission of data. There are also other handshake operations used, for example for establishing or terminating a connection, however, those have no further importance to the congestion avoidance protocol [8], [9], [11], [13].

A. Perceiving Congestion

Both partners of a connection are both sender and receiver, as both transmit packets. In the following however, we will simplify this and specify one partner as the sender, the other as the receiver. The sender keeps track of the time it takes from when a package sent until the acknowledgment is received. This is also known as *round-trip time* (RTT). Comparing the current RTT to previously measured RTTs can inform the sender about the current state of congestion. Some variances of TCP make use of this feature in their congestion avoidance algorithms, yet classic TCP does not [11], [13]. RTT also plays a very important role in the detection of packet loss, which further implies congestion. The sender side keeps track of the RTT while it is being measured and compares it to the *re-transmission timeout* (RTO) [9]. RTO time is being calculated by estimating the mean and variance of the RTT. Whenever packets are not acknowledged within this RTO interval, the TCP sender regards this packet as lost and will retransmit its data. Therefore, it is essential to TCP performance, that the RTO is being determined accurately, as a too low RTO could prompt unnecessary retransmissions and too high RTO could slow down the application TCP is serving [11], [13]. To implicitly notify TCP that congestion occurs, congested network nodes will occasionally drop packets after a certain threshold of queued packets has been exceeded. Usually, this is quite smaller than the maximum queue length, therefore avoiding actions can be taken in time [14].

B. Adjusting for Congestion

To prevent congestion altogether, packet flow inside a network would have to be conservative, i.e. a new packet isn't being sent into the network before an old packet leaves the network. A connection in this state is also being described as being *in equilibrium* [9]. Whenever a connection is in equilibrium, congestion in the network cannot be caused by this TCP connection, thus also controlling congestion. TCPs congestion avoidance algorithm can be categorized into three sub-algorithms, that all have the goal to keep the connection in equilibrium, no matter how congestion was perceived. Here, a variable called *congestion window* (cwnd) has an important role. It describes the maximum amount of packets sent during an average RTT. An additional mechanism of keeping a connection in equilibrium is, that each acknowledgment packet received by the sender opens the space for a new packet to be sent into the network [9]. The principles TCP uses to adjust for congestion are based on *Additive-Increase Multiplicative-Decrease* (AIMD) principle.

Slow-Start: The *slow-start* algorithm is used when a connection has just been established or when multiple packet losses were detected. The cwnd variable is set to one packet per estimated RTT, because no information about network congestion has been collected yet. Whenever an acknowledgment packet is being received, cwnd will increase by one packet. This doubles cwnd each RTT, resulting in an exponential growth over time, which helps to get the packet send rate to the maximum the network can handle quite quickly, contributing to almost immediately maximum network performance [9].

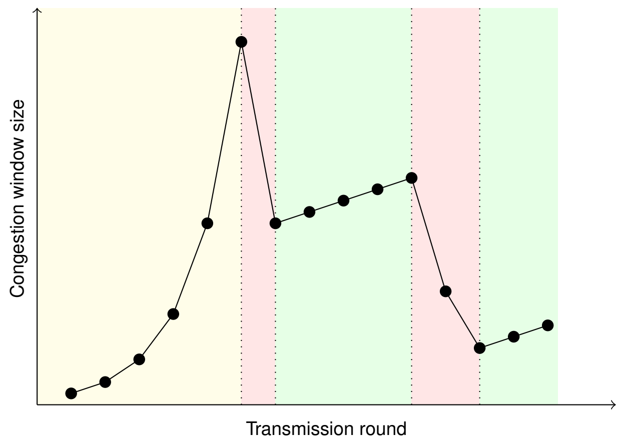

Fig. 3. TCP congestion avoidance protocols. Yellow represents the slow-start sequence, red and green symbolize detected congestion

Adaption: When the network is congested, buffers in network nodes will fill up exponentially [9]. Therefore, the network can only stabilize if the amount of packets sent decreases at least as quickly as the queues are growing. The formula [9]

$$cwnd_{i+1} \leftarrow d \cdot cwnd_i, 0 < d < 1$$

has an exponential decrease over time, thus resulting in the network recovering.
If the network isn't congested, the sender should try to raise cwnd slowly to reach the networks maximum throughput again. A multiplicative, exponential increase over time, like the slow-start algorithm isn't the best choice here. Increasing the amounts of packets sent drastically while cwnd still is high can challenge the networks capacity, resulting in heavy congestion. Therefore, the best option to raise cwnd to its limits is an additive increase [9]:

$$cwnd_{i+1} \leftarrow cwnd_i + u$$

Van Jacobson proposes in his 1992 paper about TCP congestion avoidance that d will be set to $\frac{1}{2}$ and u will be set to $\frac{1}{cwnd_i}$. Thus, cwnd can only increase by a maximum of one packet per RTT leading to a smooth approach to the networks limit [9].

IV. Network Assisted Congestion Control

The problem that comes with classic TCP is that the congestion detection algorithms are very implicit. Packet loss for example may have different causes other than network congestion, such as simple cable interference, however, the congestion avoidance algorithm will still interpret it as congestion, thus possibly resulting in unnecessary network delay. Therefore, exactly determining whether the network is congested or not is crucial to keep a network stable. Here, the network layer comes into play. Because most communication should be hidden along the path of the packet, all network

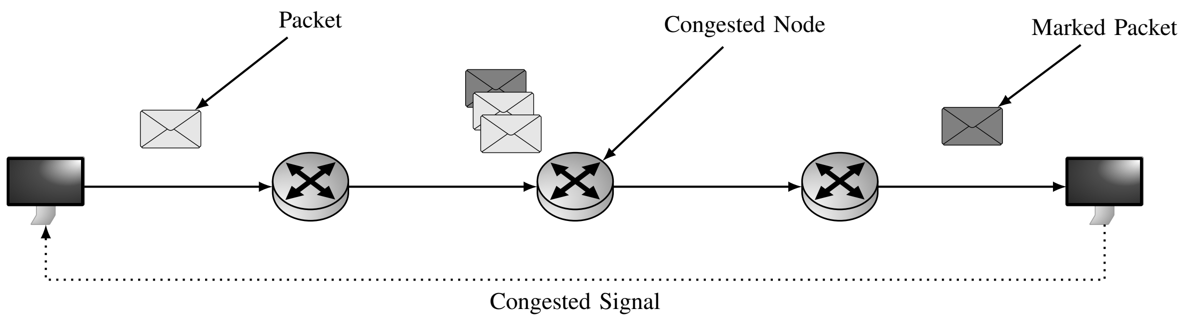

Fig. 4. Congestion Control Algorithm. A congested node marks buffered packets to describe congestion. This signal will be sent back to the original sender.

nodes will only communicate on the network layer (where IP lives), but not above (where TCP lives) [14].

In 2001 a new method was standardized to explicitly notify TCP on congestion. *Explicit Congestion Notification* (ECN) is an extension to TCP and IP, allowing end-to-end congestion notification, while also minimizing the amount of packets dropped. This is done by modifying the IP header to be able to carry one more bit - the *Congestion Experienced* flag (CE) [14]. Instead of dropping packets occasionally after the amount of queued packets exceed a threshold [Perceiving Congestion], all packets supporting ECN will now be marked (setting the CE bit in the IP header) [Figure 4]. Thus, congestion can be described explicitly resulting in better short- and long-term network performance. Finally, if a data packet has been marked, the according acknowledgment packet will also be marked to inform the sender of congestion [14], [15], [16].

To sum up so far, without ECN, classic TCP would just drive the network into congestion, and then recover from it. The network generally speaking will at least experience some jitter (fluctuating network delay). With ECN however, packets in congested nodes will be marked, explicitly informing the TCP sender that the network is congested. As a result, avoiding actions can be taken immediately. In addition, the need to retransmit packets would decrease drastically, as no packets are being dropped anymore to indicate congestion. Therefore, this central algorithm is also being called *Congestion Control* [14], [15].

A. XCP

Although TCP is a very dominant protocol on the internet, there are also other transport protocols that could benefit from end-to-end congestion notification. The paper ”Congestion Control for High Bandwidth-Delay Product Networks” generalizes the ECN proposal of 2001, creating a new protocol called *eXplicit Congestion Protocol* (XCP), while also improving upon it further by introducing new packet control concepts [17], [18].

XCP Header: Like TCP, XCP is a window-based congestion control protocol intended for the best effort. Underneath the hood however, they are very different. A major difference between both protocols is that XCP does not only inform the sender on congestion, but does also provide information about the degree of congestion. XCP can achieve this by including a *congestion header* to each packet. While the sender maintains a *cwnd* and *RTT* variable, as with TCP, those are also being communicated to network nodes via the congestion header as *H_cwnd* and *H_rtt*. In addition, the congestion header includes one more field. This *H_feedback* field is also initialized by the sender, but is the only field which can be modified by nodes along the path [17], [18].

XCP Sender: As with TCP, the XCP sender maintains a congestion window, *cwnd*, and an estimate of the round trip time, *RTT*. On packet departure, the *H_cwnd* and *H_rtt* fields will be initialized with *cwnd* and *RTT* accordingly. The initialization value of the *H_feedback* field represents a desired send rate r:

$$H\_feedback \leftarrow \frac{r \cdot RTT - cwnd}{cwnd}$$

This is an improvement to TCPs adjustment algorithms as the desired rate can be reached after one RTT.

The other task an XCP sender has is to adjust the congestion window according to the networks feedback. When s is the packet size, the congestion window variable would be adjusted by following formula:

$$cwnd \leftarrow \max(cwnd + H_feedback, s)$$

In addition to responding to direct feedback, the XCP sender still needs to handle packet loss, though this is very similar to what TCP does [17].

XCP Receiver: The XCP receiver functions exactly the same way like a TCP receiver would except for copying the (modified) congestion header from the data packet to its acknowledgment [17].

XCP Router: The main task of network nodes in XCP, or XCP routers, is to inform the XCP sender of the current state of congestion. This is done by computing feedback at every node. The objective of XCP is to prevent the queues building up to a point at which packets have to be dropped [17].

Feedback is computed by two controllers. The *Efficiency Controller* (EC) tries to maximize link utilization while minimizing queue lengths and packet drop rates. When the XCP routers link is underused, positive feedback should be sent. If the link is congested however, negative feedback should be sent in order to keep queue lengths small resulting in no packet

drops [17], [18].
To achieve efficiency and equilibrium, this feedback will be allocated to single packets as *H_feedback*. However, the EC does not decide which packets should carry the feedback. Hence, the *Fairness Controller* (FC) comes into play. As TCP, FC relies on the AIMD principle [Adjusting for Congestion]. If the EC calculated positive feedback, it will be split up between all packets equally. If the feedback is negative however, it will be divided between packets in proportion to their current send-rates, which are determined by *H_cwnd* and *H_rtt* [17], [18]. Decoupling the feedback computation allows XCP to quickly acquire the maximum send rate and achieve equilibrium, all while the network allocates bandwidth to all XCP senders equally [17].

B. Evaluation of Network Assisted Congestion Control

As you can see, including the network layer to detect congestion can have great performance improvements. 'Understanding XCP: Equilibrium and Fairness' has shown through simulations that many network topologies could benefit from XCP as it clears queues in equilibrium, thus resulting in improved network performance under load [18]. Another paper has shown that XCP outperforms ordinary TCP in nearly any environment related to average queue lengths, efficiency, and throughput [17].
However, all this performance comes at a cost. Since network nodes have to actively detect congestion, ECN, XCP, and other network assisted congestion control protocols rely on active queue management in those nodes. Although there is some hardware being produced today that meet ECNs requirements, a large part of internet hardware isn't capable of doing so. Because any of those nodes can be congested, it would be counterintuitive to run those protocols, as congestion might or might not be detected, thus resulting in performance loss [17]. This is the reason you do not see those protocols in wide area networks (WAN) too often, let alone the internet.
Nevertheless, whenever the implementation of XCP, ECN, or similar protocols is not a challenge, you can implement network assisted congestion control with ease and see great performance improvements. Especially smaller networks with high traffic could definitely benefit from those technologies.

C. DCTCP

A lot of internet services run in data centers with large amounts of computers communicating with each other, resulting in a lot of traffic inside the data center. Optimizing those data centers networking capabilities can result in an increase in application productivity. In 2010 a paper proposed *DCTCP*, a variation of TCP using ECN to create a data transportation protocol, which especially takes hardware purposed for use in those environments into account [19].
The general problem that appears in high-traffic environments is the so-called *TCP Incast* problem. It describes a drastic reduction in application throughput. To understand where TCP incast originates, you fist need to understand how most data centers are built. A typical data center contains a set of racks, each holding a large number of servers, the switches, that connect those racks, and connecting links, that connect those switches to other parts of the data center or the internet [Figure 5]. This design pattern is also known as *Partition/Aggregate* and is most commonly used in data centers. It however results in a high 'fan-in', as multiple servers (leaves) are connected to just one link (root) [19], [20]. In addition, most applications require high bandwidth, low latency connections, also enabling them to send many parallel requests to other servers. Combining those requirements with the design pattern, it becomes clear, that those switches are being challenged. Besides, many data center switches have relatively small, shared buffers. Therefore, it is common that queue lengths stay high for a prolonged period of time, resulting in a collapse of throughput [19], [20].

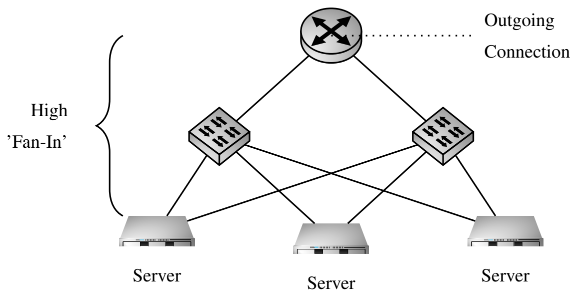

Fig. 5. Data Center High 'Fan-In'

The goal of DCTCP is to keep data center traffic constantly flowing, which means meeting the requirements applications take while keeping relatively small buffer sizes inside switches in mind. To achieve this, DCTCP implements ECN. Unlike TCP however, if a DCTCP sender registers congestion, the window size will be reduced by a fraction of the amount of marked packets it received. Therefore, excessive link under-utilization can be prevented. In addition, the DCTCP receiver will only send one ACK packet for every m consecutive data packet, where all $1, ..., m$ packets before had the same CE-flag configuration. Here, m is a variable initialized while the DCTCP connection is being established and cannot be changed without renegotiation. Therefore, fewer packets are sent through the network, but the TCP sender is still able to identify lost packets and register congestion [19].
Being able to react to congestion immediately enables DCTCP to reduce buffer overflows and timeouts, thus largely resolving incast problems. The paper proposing DCTCP has found, that their protocol increases traffic by 110% while using 90% less buffer space compared to TCP [19].

V. Conclusion

In this paper we have analyzed origins of congestion. Different factors play a role in network congestion, however especially the filling up of buffers at congested nodes will cause traffic delay. Further, we shine a light on the functionality of congestion avoidance algorithms, especially classic TCP. However, because those algorithms quite often do cause network stress rather than resolving it, we took a closer look at network assisted end-to-end congestion control. ECN, an addition to TCP, and XCP are two standardized protocols applying those principles, but haven't been very successful in wide area networks (WAN). In smaller networks, the implementation of them is easier. DCTCP, a variation of ECN, as an example, can successfully resolve many issues occurring in data centers.

REFERENCES

[1] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, 2004.

[2] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the internet," *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.

[3] C. Staff, "Bufferbloat: What's wrong with the internet?" *Communications of the ACM*, vol. 55, no. 2, pp. 40–47, 2012.

[4] C. Villamizar and C. Song, "High performance TCP in ANSNET," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 45–60, 1994.

[5] M. Allman, "Comments on Bufferbloat," *Computer Communication Review*, pp. 31–37, 2013.

[6] V. G. Cerf, "Bufferbloat and Other Internet Challenges," *IEEE Internet Computing*, vol. 18, no. 5, pp. 80–80, 2014.

[7] Y.-C. Chen and D. Towsley, "On bufferbloat and delay analysis of multipath TCP in wireless networks," in *2014 IFIP Networking Conference*. IEEE, 2014, pp. 1–9.

[8] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP throughput and loss," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 3, 2003, pp. 1744–1753 vol.3.

[9] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFc 1323, May, Tech. Rep., 1992.

[10] W. Eddy, "Transmission Control Protocol Specification," Internet Engineering Task Force, Internet-Draft draft-ietf-tcpm-rfc793bis-16, Mar. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-rfc793bis-16

[11] G. Huston, "Tcp performance," *The Internet Protocol Journal*, vol. 3, no. 2, pp. 2–24, 2000.

[12] J. Postel *et al.*, "Transmission control protocol," 1981.

[13] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 1, pp. 157–187, 1995.

[14] K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, January, Tech. Rep., 1999.

[15] K. Ramakrishnan, S. Floyd, D. Black *et al.*, "The addition of explicit congestion notification (ECN) to IP," 2001.

[16] S. Floyd, "TCP and Explicit Congestion Notification," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 5, p. 8–23, Oct. 1994. [Online]. Available: https://doi.org/10.1145/205511.205512

[17] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, 2002, pp. 89–102.

[18] S. H. Low, L. L. H. Andrew, and B. P. Wydrowski, "Understanding XCP: equilibrium and fairness," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 2, 2005, pp. 1025–1036 vol. 2.

[19] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, p. 63–74, Aug. 2010. [Online]. Available: https://doi.org/10.1145/1851275.1851192

[20] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP Incast Throughput Collapse in Datacenter Networks," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, ser. WREN '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 73–82. [Online]. Available: https://doi.org/10.1145/1592681.1592693