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

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

***Abstract*—This article presents the most common origins of network congestion, how different types of algorithms perceive and handle congestion and what obstacles they face.**

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. 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 would get, the amount of connections would grow quadratic.
Networks today consist of network nodes (routers, switches, ...) which all try to route internet packets as quick 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 face new problems with each packet sent. Therefore, the global reduction of those complications are an important task to improve the quality of all internet services. However, as it is for solving any problem, the origin of those problems must be identified first.
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 a 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 packets path (bottleneck) [Figure 1].

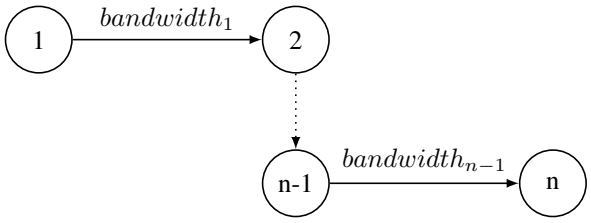

Fig. 1. Path a packet takes through the network

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

- *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 spent waiting for it 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 differ 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 as short as possible to minimize additional delay. On the other hand buffers should be as big as needed to be robust for future network changes and therefore store lots of data if needed while also keeping in mind the requirements congestion detection algorithms have in order to adjust for congestion accordingly [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 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 [1]. 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 an 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 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 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, so those protocols can adjust. 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. Thus, 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 an 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 functions, you will have to understand how TCP works. TCP lives on the transport layer. It communicates with the application layer via ports, splits up data into packets (and vice versa), and transmits those packages over the network with the use of the internet protocol, thus TCP is often also referred to as *TCP/IP*. It is the interface between network components. Each of those communication packages have different purposes, but underlay the same structure - the protocol header. The TCP header is split up into multiple segments [Figure 2][9], [10]:

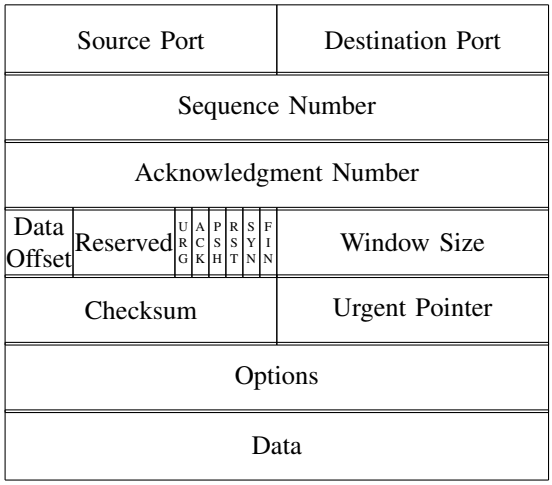

Fig. 2. TCP Header

- *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.
- TCP can set six different bit-flags in its header. The most important one is the *ACK* flag (Acknowledgement bit). It is used whenever a packet receives the destination and an acknowledgement packet has to be sent back. 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 being negotiated at the initialization of a new connection and can only be negotiated again with the *SYN* flag set. It is important to set it appropriately to prevent unnecessary packet loss or delay, as the receiver couldn't process the data in time [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], [10], [11].

A. Perceiving Congestion

Both partners of a connection are either a sender or a 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), where a large RTT can indicate some congestion along the path. Some variances of TCP make use of this feature in their congestion avoidance algorithms, yet classic TCP does not [10], [11]. RTT also

plays an very important role in the detection of packet loss, which further imply congestion. The sender side keeps track of the RTT while it is being measured and compares it to the *retransmission 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 intervall, the package is being perceived as lost and will be retransmitted. 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 [10], [11]. 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 the maximum queue length, therefore avoiding actions can be taken in time [12].
Note however, a high RTT can also have other causes than packet loss or congestion. A short term drastic decrease in the receivers processing performance can lead to packets being buffered at the receivers end for a prolonged period of time, thus resulting in a higher RTT [10].

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 packages send during a interval. An additional mechanism of keeping a connection in equilibrium is, that each acknowledgement package received by the sender opens the space for a new package 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 package per estimated RTT, because no information about network congestion has been collected yet. Whenever an acknowledgment package is being received, cwnd will increase by one package. This exponential growth in packages helps to get the packet send rate to the maximum the network can handle quite quickly, resulting in almost immediately maximum network performance [9].

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]

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

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 result in the network being 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].

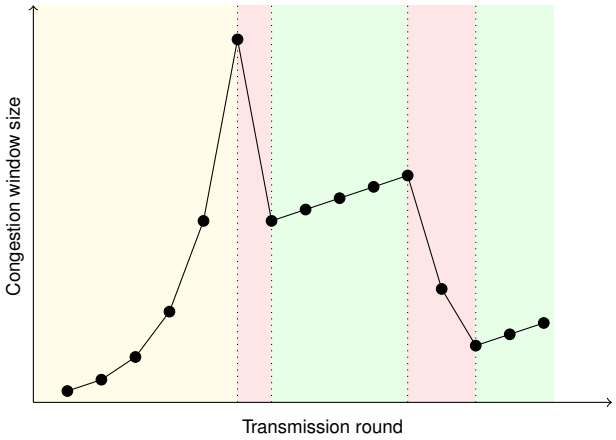

Fig. 3. TCP congestion avoidance protocols. Yellow represents the slow-start sequence, red and green symbolize detected congestion

IV. Network Assisted Congestion Control

The problem that comes with classic TCP is that the congestion detection algorithms are very implicit. As already mentioned, packet loss for example may have different causes other than network congestion, 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 packets path, all network nodes will only communicate on the internet layer (where IP lives), but not above (where TCP lives) [12].
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

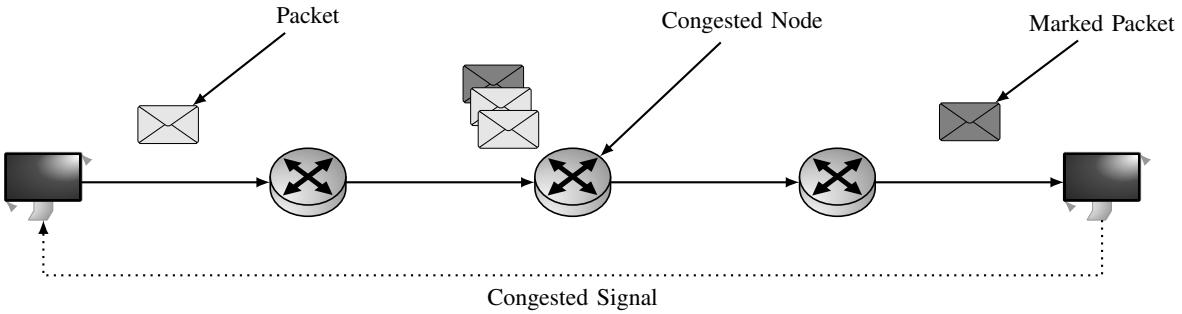

Fig. 4. TCP/IP with Explicit Congestion Notification

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) [12]. 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). Thus, congestion can be described explicitly resulting in better short- and long-term network performance. Also, if a data packet has been market, the according acknowledgment package will also be marked [12], [13], [14].

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. Therefore, we also speak of *Congestion Control* [12], [13].

A. XCP

Although TCP is a very dominant protocol in the internet, there are also other transport protocols which 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 [15], [16].

XCP Header: Like TCP, XCP is a window-based congestion control protocol indented for best traffic. 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 [15], [16].

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 package loss, though this is very similar to what TCP does [15].

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 [15].

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 [15].

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, a positive feedback should be sent. If the link is congested however, a negative feedback should be sent in order to keep queue lengths small resulting in no packet drops [15], [16].

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 a positive feedback, it will be split up between all packages 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* [15], [16].
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 [15].

B. Evaluation of Network Assisted Congestion Control

As you can see, including the network layer to detect 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 [16]. Another paper has shown that XCP outperforms ordinary TCP in nearly any environment related to average queue lengths, efficiency and throughput [15].
However, all those performance comes with 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 most hardware being produced today meet those 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.
Nevertheless, whenever 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 computers communicating with each other, resulting in a lots of traffic inside the data center. Optimizing those data centers networking capabilities can result in an increase of quality of services, which they are hosting. PAPER OF *DCTCP* proposes a variation of TCP using ECN to create a data transportation protocol, which specially takes hardware purposed for use those environments into account.

V. Conclusion

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

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

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

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

[14] 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

[15] 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.

[16] 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.