

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

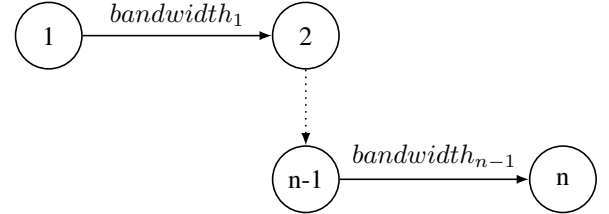


Fig. 1. Path a packet takes through the network

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].

$$Rate_{max} = \min\{bandwidth_i \mid i \in \{1, \dots, n-1\}\}$$

- *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

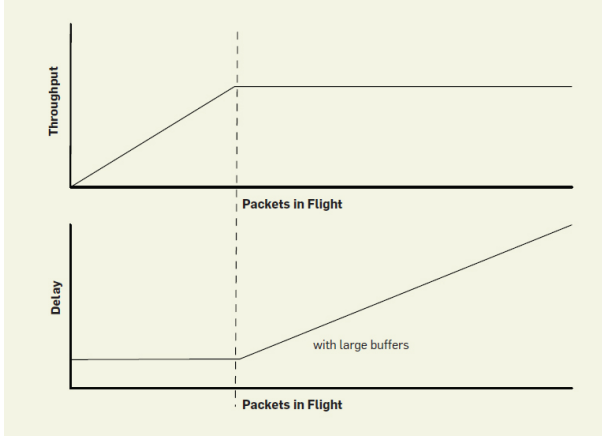


Fig. 2. Relationship between throughput and delay [2]

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 [Figure 2][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.

Source Port				Destination Port			
Sequence Number							
Acknowledgment Number							
Data Offset	Reserved	URG	ACK	P	S	R	Window Size
		R	S	T	T	S	
Checksum				Urgent Pointer			
Options							
Data							

Fig. 3. TCP Header

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 3][9], [10]:

- *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 detected 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]. 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].

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

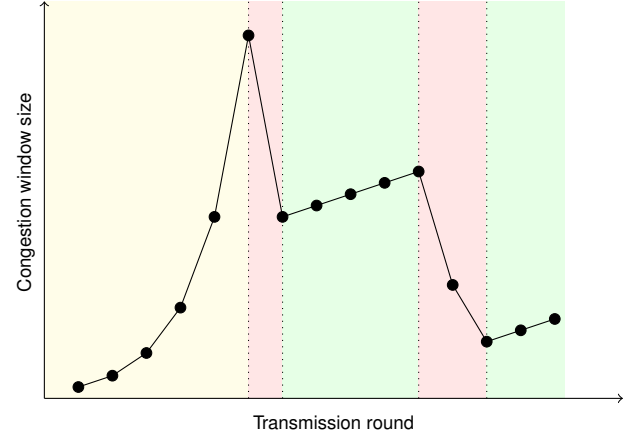


Fig. 4. TCP congestion avoidance protocols. Yellow represents the slow-start sequence, red and green symbolize detected congestion

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]. Slow-start will only increase if cwnd

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]

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

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]:

$$cwnd_{i+1} = cwnd_i + u$$

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

Congestion Control anhand von ECN (TCPIP-Erweiterung) erklären Bit im IP Header setzen + wie TCP damit umgeht Vorteile (genaue Maßnahmen) und Nachteile (erforderliche Hardware) XCP (Verallgemeinerung von ECN)

DCTCP

DCTCP + Warum nur in Data Center verwendet

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.