Overview of TCP Congestion Control Algorithms

Moritz Geist, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: moritz.geist@tum.de, jaeger@net.in.tum.de

***Abstract*—The current set of congestion control algorithms is split into three primary groups regarding their function and can easily be categorized and therefore characterized. This paper compares these different classes in compliance with their respective advantages and disadvantages. Each algorithm on its own is well researched, however, it is difficult to predict how they perform when used together in the Internet due to the unpredictable behaviour of it.**

***Index Terms*—tcp, congestion control algorithm, measurement, high-speed networks**

1. Introduction

With the world being more and more connected through the Internet, the underlying network has to work increasingly efficient to achieve a high-performing and stable connection all the time. That is a very difficult goal to achieve without the help of the transmitting computers. If every computer connected to the Internet would just send packets as fast as possible, the slowest links or most utilized networks would get overloaded to the point of routers dropping packets instead of passing them on to the next node. This leads to a severe performance hit for Internet applications due to the whole stream having to wait for the packet to be retransmitted. To avoid this and keep a high throughput, while not losing packets, computer scientists have come up with several congestion avoidance algorithms. These algorithms work by controlling the size of the congestion window. The congestion window limits the number of packets that can be inflight, meaning waiting for acknowledgement, at any time and is created for every TCP connection. As long as there are less packets inflight than the size of the congestion window, new packets will be sent out and transmitted. The packets get removed from the congestion window as soon as they are acknowledged, which allows for the next packet to be sent. By this definition, the optimal congestion window size is equal to the Bandwith Delay Product (BDP). The BDP is calculated using $BDP = bandwith \cdot RTT$. The bandwith to be used is the total usable bandwith of the slowest link on the path, and RTT is the total round trip time of the stream. This is due to the nature of routers between the sender and the receiver: Internally, they operate a packet queue that incoming packets will be appended on. The packet at the head of the queue will be processed and transmitted to the next node. If it happens that more packets come in than the number of packets that get sent out, the packet queue will grow until a certain limit. If the limit is reached, packets will be dropped until there is new space in the queue. If the sender just continues to send out packets as fast as possible, all packets would get dropped on that link. Instead, the congestion control algorithm reduces the congestion windows size until no more packets are lost on their way. The result of a higher congestion window than BDP is therefore a queue filling up at the slowest link, while a smaller congestion window decreases the size of the queue gradually.
The primary difference in congestion avoidance algorithms is how they detect an overloaded link in between them and how they increase and decrease the congestion window. This paper will compare different methods of detecting and handling the size based on some existing algorithms.
This paper will first explain the different techniques on how a congestion control algorithm can operate in Section 2, followed by examples that highlight each of the different approaches in Section 3. In Section 4, the algorithms will be discussed in how they perform compared to each other and especially while active as parallel streams in a network.

2. Background

In this section, we compare how different congestion avoidance algorithms detect an overloaded link on their path. While it is most common to utilize only one of these three strategies, some algorithms use combinations of them. As Lefteris Mamatas et al. specify in [1], the algorithms can be classified into three distinct groups: black box algorithms that do not rely on any state information about the network and only rely on binary feedback; gray box algorithms that do active measurements to "estimate available bandwidth, level of contention or even the temporary characteristics of congestion" [1]. Third, green box algorithms do have exact knowledge about the state of every (or most) part of the network, either by being implemented in every part of it or every link reporting its status to the sender.

2.1. Loss-based Algorithms

One of the most common ways of detecting an overloaded link is by reacting to packet loss. This is considered a black box method, as it only relies on the binary input of a packet being lost or not. In case a packet is not acknowledged after a certain time, called the retransmission timeout, or after receiving three duplicate ACKs [2], it is considered missing and will be retransmitted. This also indicates to the congestion avoidance algorithm to

doi: 10.2313/NET-2019-06-1_03

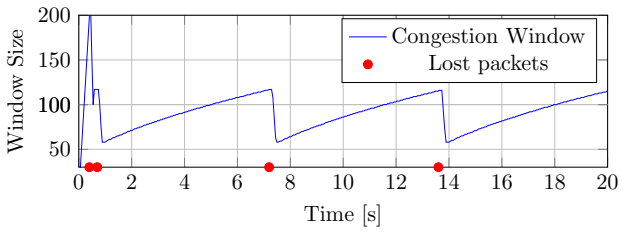

Figure 1. Congestion window resize after a lost packet in TCP Reno

shrink the congestion window, so less packets are sent out. In the most simple algorithm, called TCP Reno (seen in Figure 1), the congestion control begins with a phase called "slow-start" up to $t = 1s$ [3]. In that, it the window size is increased by one for each acknowledged packet resulting in an exponential growth, until the first lost packet is registered. After that, the congestion window is halved on loss, and will be increased steadily, but in a linear way. This results in a sawtooth-like graph for the congestion window size like in Figure 1. Loss-based congestion avoidance algorithms only need to be implemented by the sender, making them exceptionally easy to deploy. The receiver, who also is a sender by at least sending out acknowledgements, can use a totally different congestion avoidance algorithm. The problem with these algorithms is that packets can get lost for reasons other than an overloaded link, for example, if an actually broken, or a less reliable link is used on the way. This results not only in a bad performance due to retransmits, also the congestion window is decreased without any need making it smaller than the BDP and therefore decrease effective throughput. Even if packets are only lost due to overloaded links, these algorithms can never be perfect: The window size is increased as long as there is no packet loss, which in turn means that there always will be at least one lost packet every so often, as seen in Figure 1. This can lead to problems with applications that require near-real-time communications, like Voice-Over-Ip or online games. Algorithms based on packet loss are therefore the easiest to implement, while also theoretically most limited considering accurate functionality.

2.2. Delay-based Algorithms

Another way of detecting an overloaded link is by measuring the delay in which acknowledgements arrive, also known as the round trip time (RTT). By the criterion listed in Section 2, these algorithms are grey box algorithms, as they use more advanced measurements to examine and monitor the status of the network. Especially by monitoring changes in RTT, these algorithms can react to a congestion earlier than purely loss-based algorithms, as most of the time the RTT increases gradually before a packet is actually lost. The queue fills up like explained in Section 1 when the congestion window is bigger than the BDP. A nonempty queue means that packets have to wait in line to be sent on, leading to a higher RTT. Only when the queue size reaches he maximum buffer size, packets are dropped instead of being queued. This has the following advantages: Delay-based algorithms can react sooner to congestion, maybe even before the first packet is lost at all, which can have a positive impact on the performance of some applications. Also, instead of having to cut down the congestion window in half, it can gradually shrink the congestion window relative to measured increase in RTT. This can lead to an improvement in throughput compared to other black box algorithms in isolated environments. A very basic implementation of a delay-based algorithm is TCP Vegas [4]. An example of this behaviour can be seen in Figure 2. As soon as the network gets congested at *t=10*, the buffer of the overloaded link starts to fill up, resulting in the average RTT increasing. The sender reduces the congestion window to prevent any packet losses.

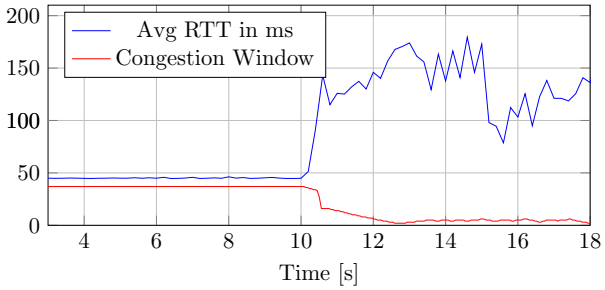

Figure 2. Decrease of Congestion Window Size

While a delay-based algorithm is alone in a network, it is able to adjust its congestion window to a perfect size: Maximum throughput combined with no packet loss and minimum RTT. It gets difficult as soon as another connection uses the same link that is more aggressive, like a loss-based algorithm mentioned in Section 2.1. As these will enforce one or more lost packets from time to time, this affects the delay-based stream as well because of the increased queueing delay, leading to it reducing the congestion window to an absolute minimum. Also, delay-based algorithms can react to loss similar to loss-based algorithms as well, as seen in Section 3.2. How the detection technologies work together will be described in Section 4.

2.3. Signal-based algorithms

As the definition in [1] states, with green boxes "the network communicates its state to the transport layer". This is achieved through signal bits, with which an overloaded link notifies the sender of its state. The actual implementation of this can vary. The TCP protocol header contains six unused bits that are reserved for future use as well as an option field that can carry more complex information [5].The advantage of this algorithms is obvious: With complete knowledge of the status of the network, it becomes a matter of algorithmic design of how the sender should adjust its congestion windows size. It is even possible to prioritize different streams of data without exceeding the load on a link, making sure it does never get actually overloaded and incur high delays or drop packets. A shortcoming of this method is illustrated in Figure 3. The link between the router and the receiver is slower than the link between the sender and the router, which means it will be overloaded at some point. The router detects this as soon as the internal buffer starts

doi: 10.2313/NET-2019-06-1_03

to fill up, and will set signal-bits or options in the next packet it forwards to the receiver accordingly. The receiver will then acknowledge the packet and adopt the extra information in the corresponding acknowledge-packet.

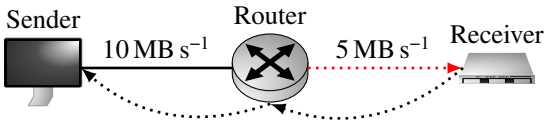

Figure 3. Path of the signal

As visible in Figure 3, the information about the increased load needs to travel at least three hops, depending on the network layout. It follows the dotted lines back to the sender. Therefore, the information takes at least half of one round trip time to reach the sender. The big disadvantage of signal-based algorithms or green box algorithms is that they need to be deployed on all parts of the network, including the receiver and every router. This makes it practically impossible to use them in the open Internet, so they are only useful in closed networks like offices or data centers where each of the network components is controlled by a single instance. Loss-based 2.1 and delay-based 2.2 algorithms do not require this.

3. Implementation

This section will showcase some more- and less popular congestion avoidance algorithms and highlight their benefits and shortcomings. It will cover all of the before mentioned types of detection from Section 2.

3.1. TCP Cubic

TCP Cubic is the current default congestion avoidance algorithm in the current Linux kernel, which makes it one of the most used algorithms. It has been developed by Sangtae Ha et al. in 2008 and the specification is made available here [6].
TCP Cubic is completely loss-based and therefore a black box algorithm. It can be viewed as an improvement over TCP Reno described in Section 2.1 and Figure 1. After the same slow-start-phase that is present in TCP Reno it behaves similarly when receiving a lost packet. The main and only difference is how it increases the congestion window. Instead of increasing it with a constant (or at least linear depending on the current RTT) function it splits the increase function into two phases that are illustrated in Figure 4: First, it will increase the congestion window based on a concave cubic function (right curved) until it reaches a value called W_max at $t = 8$. This is the size the congestion window had when the last congestion event (packet loss) occurred. After reaching that point again, the function begins to increase slowly at first, but with increasing speed later on (convex, left curved). This method of growth causes the congestion window and therefore sending rate to stay close to the last known highest value as long as possible, while still be able to go beyond that. Consequently, TCP Cubic behaves very similar to TCP Reno while being able to faster recover from a loss and slower run into the next one.

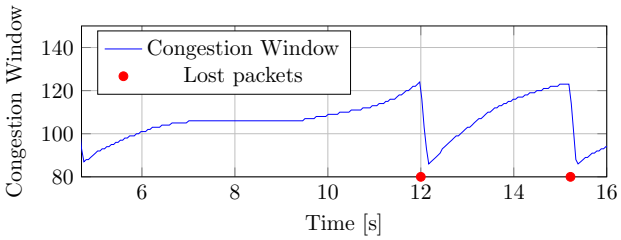

Figure 4. TCP Cubic Congestion Window Size

3.2. LEDBAT

Low Extra Delay Background Transport (LEDBAT) is a delay-based congestion control algorithm that has been technically documented in 2012 [7]. It can be seen as a grey box algorithm, as it measures the time a packet travels from sender to receiver, not just the round trip time. To do this, a timestamp is appended to every outgoing packet that the receiver then subtracts from his local time and responds the one-way delay to the original sender using the acknowledgement packet. The sending side then considers the difference in delay over time, so clocks do not need to be synchronized. LEDBAT purposefully only uses the one-way delay for its calculations, as it is designed for primarily one way bulk-transfer applications like file-sharing and software updates. Its goal is to reduce the impact on contending other streams while still being as fast as possible. As seen in [7], LEDBAT is, in general, less aggressive than TCP Reno or Cubic due to it also halving the congestion window on loss in addition to shrinking with increasing delay. The ramp-up function is never faster than TCP Reno, therefore it will not interfere too much with other traffic. As a result, LEDBAT is very useful for data transfers that do not require real-time information processing but instead just need to get done in some amount of time. Currently LEDBAT is used by the BitTorrent system [8] as well as for updates in some operating systems.

3.3. TCP Westwood

TCP Westwood (and TCP Westwood+) is a combination of a loss- and delay-based congestion control algorithm [9]. They have been developed to improve efficiency in paths with a large bandwith-delay product and a potential packet loss such as long wireless links. In general, it behaves similarly to TCP Reno in that it features equal slow-start and congestion avoidance phases. The difference is how it handles congestion events or packet losses. Instead of halving the congestion window, Westwood uses an algorithm to estimate the real end-to-end bandwidth with which it then adaptively sets the slow-start threshold and congestion window. Due to the bandwidth esitamtion Westwood works fine in a network with more streams, unlike TCP Vegas as will be explained in Section 4. TCP Westwood+ works similar, but employs a different algorithm to estimate bandwidth, as the original algorithm was found to be flawed [10] due to compression of acknowledgement packets.

doi: 10.2313/NET-2019-06-1_03

3.4. TCP MaxNet

TCP MaxNet is a green-box algorithm from 2002 that features active feedback about the status of the network to the sender [11], [12], [13]. The routers on the way each calculate a price using an active queue management algorithm (AQM). These calculations factor in a demand function, the next links capacity and utilization. Given equal demand functions, there is max-min fairness [11]. By scaling this function, it is possible to prioitize a stream (weighted fairness). The calculated price is then transported back to the sender along the links using the max() Function 1 with p_i being the individual price value of the router at position i.

$$price = \max_{p_i} \tag{1}$$

A detailed definition of the price can be seen in [13]. The client will then adjust its congestion window according to the determined price, so the effective sending rate adjusts to the slowest link in the network. It therefore is easy to define and predict how the network behaves, as long as there are no competing streams that do not use TCP MaxNet.

4. Evaluation

An important consideration when developing a new congestion control algorithm is how it behaves in real-world scenarios, where overloaded links are hit with different streams at once, and therefore with several different rates, and different congestion avoidance mechanisms. For example, a delay-based stream will reduce its congestion window way before a competing loss-based stream observes a lost packet. An example of this is seen in Figure 2, where the congestion window of a TCP Vegas stream is visible. TCP Vegas is solely delay-based and shrinks the congestion window when the RTT increases [14]. The effect can be seen by also measuring the throughput at the congested link, which is plotted in Figure 5, showing the same timeframe as in Figure 2. As soon as the Reno stream starts at $t = 10$, it immediately congests the link in the slow-start phase, to which TCP Vegas reacts with reducing the congestion window drastically. Reno will then keep the link congested regularly with a congestion window similar to Figure 1, leading to TCP Vegas never increasing its congestion window again.

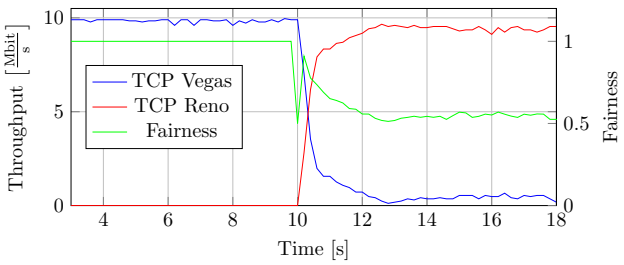

Figure 5. Throughput at the congested link

This can be calculated using a fairness measurement, a value that describes how well the total throughput is shared among all streams. One way to calculate such a value is by using an equation developed by Raj Jain [15]. It produces values ranging from $\frac{1}{n}$ (not fair), where n is the number of streams, to 1 (fair). For the above example, the index is of value 1 until $t = 10$, and then drops to about 0.5. Therefore, this example showcases a worst-case scenario in terms of fairness. Other delay-based algorithms try to improve on that shortcoming, for example, Westwood and Westwood+ behave better in the same scenario with fairness values of around 0.8 on average, while still being outperformed by TCP Reno in terms of effective throughput. Generally, tt would be best if everyone would be using the same algorithm, just like a green-box algorithm like MaxNet (see Section 3.4) encourages. These do then accomplish MaxMin fairness, meaning equal share for every stream with a fully saturated but not overloaded link.

5. Related Work

The idea of increasing the overall efficiency of the Internet is quite interesting, therefore a lot of people already tried their best at developing a new, best congestion control algorithm to the point that a "Yet Another Highspeed TCP"-algorithm "YeAH-TCP" [16] exists. Other newly introduced alforithms include BBR developed by Neal Cardwell et al. that tries to determine the actual level of congestion using active probing of the network [17] and PCC Vivace by Mo Dong et al that utilizes machine learning to improve network efficiency [18]. There is also research that compares the performance of algorithms like L. Grieco does in [19]. The paper *The macroscopic behavior of the TCP congestion avoidance algorithm.* [20] explains how TCP congestions can affect the real Internet.

6. Conclusion and Future Work

While there are many congestion control algorithms that each have their pros and cons, it is very difficult to have them existing next to each other, unless it is the design goal (like with LEDBAT in section 3.2) to have a lower-priority stream. The existing methods for judging fairness work well to calculate the fairness afterwards, but with the chaotic nature of the Internet it is very difficult to predict the behaviuor of the packets. All the algorithms more or less expect that every packet travels through the same links every time, which is not guaranteed in any way. There are even projects that try to increase throughput and failure handling by explicitly using different interfaces and links like MultiPath TCP [21]. In future work, it would be interesting to look at the performance if algorithms in changing network environments and how fast they adapt to big variations in maximum throughput and RTT.

References

[1] L. Mamatas, T. Harks, and V. Tsaoussidis, “Approaches to congestion control in packet networks,” *Journal of Internet Engineering*, vol. 1, no. 1, 2007.

[2] W. R. Stevens, “Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms,” 1997.

[3] M. Allman, V. Paxson, and E. Blanton, “Tcp congestion control,” Tech. Rep., 2009.

[4] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*. ACM, 1994, vol. 24, no. 4.

doi: 10.2313/NET-2019-06-1_03

[5] P. SPECIFICATION, "Transmission control protocol," 1981.

[6] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[7] M. Kuehlewind, G. Hazel, S. Shalunov, and J. Iyengar, "Low extra delay background transport (ledbat)," 2012.

[8] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "Ledbat: the new bittorrent congestion control protocol," in *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*. IEEE, 2010, pp. 1–6.

[9] L. Grieco and S. Mascolo, "End-to-end bandwidth estimation algorithms for westwood tcp congestion control," in *Information Technology Interfaces, 2003. ITI 2003. Proceedings of the 25th International Conference on*. IEEE, 2003, pp. 563–568.

[10] R. Ferorelli, L. A. Grieco, S. Mascolo, G. Piscitelli, and P. Camarda, "Live internet measurements using westwood+ tcp congestion control," in *GLOBECOM*, vol. 2, 2002, pp. 2583–2587.

[11] B. Wydrowski and M. Zukerman, "Maxnet: a congestion control architecture," *IEEE Communications Letters*, vol. 6, no. 11, pp. 512–514, 2002.

[12] B. Wydrowski, L. L. Andrew, and M. Zukerman, "Maxnet: A congestion control architecture for scalable networks," *IEEE Communications Letters*, vol. 7, no. 10, pp. 511–513, 2003.

[13] L. L. Andrew, K. Jacobsson, S. H. Low, M. Suchara, R. Witt, and B. P. Wydrowski, "Maxnet: Theory and implementation," *WAN-in-Lab project, pp1-11*, 2006.

[14] R. J. La, J. Walrand, and V. Anantharam, *Issues in TCP vegas*. Electronics Research Laboratory, College of Engineering, University of California, 1999.

[15] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[16] A. Baiocchi, A. P. Castellani, and F. Vacirca, "Yeah-tcp: yet another highspeed tcp," in *Proc. PFLDnet*, vol. 7, 2007, pp. 37–42.

[17] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[18] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance." in *NSDI*, vol. 1, no. 2.3, 2015, p. 2.

[19] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 25–38, 2004.

[20] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[21] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 266–277.