Pacing TCP Traffic and its Implementation in Modern Congestion Control

Abstract—**Modern networks handle large amounts of traffic. Often congestion occurs because more traffic arrives at an active network node than it can forward. Thus controlling this congestion is important. Luckily TCP implements standardised algorithms to do so. But most of this congestion control is purely loss-based and struggles in networks with very small buffers of a few kilobytes in size. For advances in network performance it might be essential to make TCP ready for these networks. Both fast, but difficult to scale, SRAM and optical buffers restrict networks in its available buffer size. Pacing can actually help to make bursty loss-based congestion control perform better in these types of networks. New congestion control algorithms like BBR use pacing and therefore perform better in some setting than loss-based algorithms.**

I. Introduction

Router forward packages and have to buffer these if congestion occurs. Therefore router buffer are an essential part to make networks fast and reliable. But it has been shown that the size of these buffers can change the reliability and resource efficiency greatly[1], [2]. By now there are many different approaches to size buffers which mostly depend on certain circumstances to be at an advantage. But it is also important to mention that some buffer sizes create problems. Large buffers for example will be constantly filled by most loss based congestion control algorithms and thus create large delays. On the other side are networks with very small buffers which might keep delays small but increase packet loss drastically.

Buffer sizes can thus greatly impact network performance. But there are techniques like new more sophisticated congestion control algorithms and pacing which can help to mitigate the negative effects a certain buffer size has. A particularly interesting technique, that will be discussed in more detail in this paper, is pacing and congestion control that implements it. It basically sets a certain rate at which packets are send into the network while most loss based congestion control algorithms usually send packets in burst whenever possible. To base congestion control algorithms on RRTs and available bandwidth like BBR, while not particularly new, has interesting performance measurements in experiments. Therefore we will take a closer look at these. Fistly we will recap how loss-based TCP variants deal with congestion. Secondly we will give an overview on the implication different buffer sizes have. After that we will look into pacing and how it can make loss-based congestion control algorihtms suitable for very small buffered networks. Finally we use BBR as an example for congestion control implementing some sort of pacing and compare it to CUBIC.

II. Background

TCP implements congestion control algorithms to make sure that bottlenecks dont lead to an exceding amount of retransmitions. By today there many congestion control algorithms with different approaches. Three widely discussed loss-based congestion control (CC) algorithms are Reno [3], NewReno[4] and Cubic[5]. The following section will give a broad overview on how TCP works and how it interacts with buffers.

A. Overwiew over TCP

TCP is used for the reliable transfer of data over a network. Therefore TCP uses Acknowlegements (ACK). For every packet that reaches the receiver it has to send an ACK to the sender to confirm its arrival. The sender keeps track of outstanding packets through the TCP window and it also calculates an amout of packets it is allowed to send called the congestion window (cwnd). Besides this the receiver can advertise its own window (rwnd). The smaller one of cwnd and rwnd, if rwnd exists, limits growth of TCPs window. During slow-start phase of CC the sender will increase its cwnd by one packet of every ACK received as long as none are missing. This effectively doubles the congestion window every round-trip time (RTT). TCP exchanges slow-start for congestion avoidance if cwnd exceedes a certain threshold. In the latter phase cwnd is increased less aggressively but in a different way for each of the CC algorithms. Cubic increases its cwnd by a cubic function instead of the linear way Reno does it, for example. When loss occurs cwnd is reduced and the sender waits until it is allowed to send new packets. Independent of the current phase the sender will send as many packets as necessary, when cwnd is updated, to match its window to cwnd. This fact leads to the known burstiness of TCP.

What has been explained in the previous paragraph is the standardised approach to CC, it is purely loss-based. This means it only takes packet loss as an indication for congestion and tries not to overload the network that way. This type of congestion control is currently the most widely used. In contrast to this approach there is CC that uses delay. Two examples for this are TCP Vegas and BBR. Both try to estimate the appropriate bandwith available for the according flow by monitoring RRTs. We will go into more detail how it works and when BBR improves performance in section IV. But before we sum up problems with loss-based CC and how pacing these algorithms can improve their performance.

B. Interaction with buffers

Routers have buffers to queue packets if instant forwarding is not possible. One example would be that bursts arrive at the router which can not forward all packets at once because of a bottleneck. Appropriate buffer sizes have been discussed with different results. In this study by Villamizar and Song [6] from 1994 the so called bandwidth-delay product (BDP) has been proposed. Buffers sized by the BDP are usually very large compared to what newer studies suggest but have been widely used.

TCP increases cwnd to make use of all the available bandwidth and will eventually fill even the largest buffers over time. These large buffers can hold many packets if necessary but when its capacity is depleted it will drop all further arriving packets. Nonetheless large buffers can help to reduce packet loss in case of short term burst of packet arrivals [2] and it will also keep bottleneck links from idling in a single flow case. If some packets are dropped the routers can still send these over the bottleneck while the sender waits for its TCP window to fall below cwnd again.

For TCP to recognise drops all currently buffered packets have to be send and acknowleged. This additional queueing delay makes TCP realise the drop much later than it would have with a smaller buffer. Thus it increases cwnd further which leads to even more congestion and keeps latencies high [7]. So large buffers have considerable downsides which small buffers can address.

So called small buffers(99% less than BDP) have been shown to have very little negative impact on throughput [2]. At least that is when more than 250-300 flows pass the router because then flows are desynchronised and a bottlenecks bandwidth can be fully utilised. Flow synchronisation means that flows increase and reduce their cwnd at the same time. This can happen if multiple flows lose packets due to full buffers at once, but it occurs less and less the more flows share a link. Furthermore latency is reduced, because of shorter queueing delays, but packet loss increases. Packet loss happens due to bursts, requiring buffers to hold more packets at once than it can handle, or due to flows overestimating the available bandwith over time, filling the buffer completely. Especially with samller buffers this happens more often.

This makes it clear that buffer sizing is allways a weighting between low latency and high throughput/low loss rate. Especially for the use of SRAM in routers and all optical networks this requires attention [1], [8]. SRAM could lower access times to router buffers but it is not possible to efficiently integrate enough of those memory chips to create large buffers. All optical packet switched networks have a similar restriction. Optical buffers can only holt around 20 packets at a time [9], far less than 1% of BDP and sometimes refered to as tiny buffers. For networks like this the bursty nature of TCP leads to degraded throughput and a higher lossrate. To counteract these shortcommings of loss-based CC, pacing can be used.

III. Paced TCP

To reduce TCPs bursty nature and better deal with its buffer filling behavior pacing might be a promising approach. In general there are 2 different implementations of pacing. Either pacing will be done directly by the host that sends data into the network or it is done by routers. While the first, host pacing, has to be individually used by every client sending data into the network, the latter, edge pacing [10], could be deployed to routers by whoever administrates the network.

A. How does it work?

Every host uses some sort of CC that determines its cwnd. But in all previously mentioned loss-based CC algorithms bursts occur frequently. TCP already has a mechanism called ACK-clocking that should pace its sending rate. It builds upon that the sender only transmits packets when new ACKs arrive. Since packet arrivals at the receiver are spaced out by slower transmission over a bottleneck, ACKs should be spaced out aswell. But this often does not suffice because of ACK-compression, when ACKs are buffered and thus lose their spacing. Therefore spacing has to be added at the sender side of a host using loss-based CC. For this a process is needed that handles MTU sized packets right befor they are given to the NIC. With $\frac{RTT}{CWND}$ the time between the sending of every packet can be calculated[11]. This gives the amount of time the packets are spaced out and it will be recalculated every RRT.

On the other side there is edge pacing done by edge routers. Edge routers are those which connect smaller networks or access networks to a larger core network. As smaller buffers can be advantagous in core networks for lower queueing delays, they lead to higher packet loss. This can be partially countered by pacing. As described by Gharakheili et al. [10] when burst arrive at the edge routers the packets are not immediatley forwareded. Instead the edge routers have a delay d. If that delay is 0 all packets are forwarded as they arrive. Otherwise if the delay is larger, packets may be held back and will be send over time, so that no packet is delayed longer than d allows.

B. Performance of paced TCP

Non paced and loss-based TCP perform very differently depending on the setting. On the one hand TCP loses responsiveness in networks with large buffers and can lead to heavy congestion. On the other hand it has low latencies but large amounts of packet drops in small buffered networks because of its bursty nature. This brings us to pacing and how it performs better or worse than non paced TCP variants.

We will begin with a wide spread setting: Larger buffers of size up to the BDP. In this case pacing TCP does not lead to better performance. Overall, TCP Reno achieves high throughput in networks with large buffers and in networks with small buffers but many concurrent flows [2]. However in the latter case paced TCP variants can achive a stable link utilisation of little above 80% without depending on the amount of flows [11]. While paced TCP has 80% link utilisation with 10 flows, non paced TCP has below 10% link utilisation. Admittedly this is a very limited usecase. Nowadays with modern networks handling thousands of TCP flows at a time non paced Reno outerperformes its paced variant be achieving nearly 100% link utilisation.

On the other side, when buffering is limited to only a few packets like in optical packet switched networks, pacing can yield higher throughput. For the following results evaluated by Enachescu et al. [12] a network has been used in which hosts send their traffic over access links into a core network. Even if the core is overprovisioned and the combined traffic of all access links cannot excede the core links speed, non paced TCP Reno struggles to achieve high link utilisation. With 10 packets of buffering throughput of non paced TCP is ca. 70% lower than that of paced TCP. This is because if bursts of packets arrive at the core router simultaneously, packet loss occurs due to its limited buffering capability. Pacing counteracts this. If packets are spread out it is less likely that many packets arrive at once. This way loss occurs less frequently and cwnd is less often reduced leading to higher throughput. However in case the core is underprovisiond, because all hosts combined send more traffic than it can handle, throughput of Paced TCP degrades aswell. This is because the spacing introduced at the hosts is lost by buffering and simultaneous packet arrivals. It is to mention that these findings seem to persist for arbitraty amounts of concurrent flows. Interestingly, spacing out packets, just like by using host pacing, can be achieved by reducing access links speed. In a network where traffic from access links can only be slow, there naturally is time between every inflight packet. This is where edge pacing comes into play.

Although it is restricted to a specific network topology it can help to make traffic smooth and evenly spread out in the networks core. In [10] all simulations are conducted on a core network, similar to that in the previous experiments, in which traffic is injected by end hosts over an edge router. The core router have very small buffers of up to 30 KB and the bottleneck lies at the link connecting the two core router over which all traffic travels. This means that again the cumulative amount of traffic entering the core network over access links excedes the links capacity connecting the two core router. The conducted experiments show that edge pacing performs 13% better in terms of throughput than non paced Reno and host paced traffic for core router buffers of 5-15 KB size and 1000 long-lived TCP flows. For smaller numbers(100) of flows however host pacing yields better results. In one simulation edge pacing performes even 27% better than unpaced TCP. This is the case when all links operated at 100 mbit/s and the core router buffer was 5 KB large. Edge pacing outperformed host pacing aswell which shows how efficient edge pacing can be in scenarios with large bottlenecks. For short-lived TCP flows a simulation compareable to the first one described, from Gharakheili et al. [10], indicates that edge pacing can improve throughput substancially. Another interesting result of this particular study is that edge pacing does not suffer from unfairness to unpaced flows like host pacing. Flows that are host paced can loss up to 10% on goodput when competing with unpaced Reno flows. While this does not impact overall link utilisation, it puts pacing hosts at a disatvantage which makes partial implementation of host pacing difficult. Contrary to that, edge pacing does not reduce the performance of its flows which means that it can be partially deloyed into a network to gradually increase performance.
To conclude this section it becomes clear, that pacing has some special usecases but can not universally improve a networks throughput and loss rate. Yet it can help to make TCP fit for the use of either SRAM or optical packet switching. Furthermore it can be a useful tool to new CC algorithms like BBR.

IV. Congestion Control Algorithm BBR

Congestion Control protocol, a key-function of the transport layer, have until now already made a lot of changes in wireless and wire-lined networks. Congestion control is needed to achieve high network utilization for multiple flows [13] and to prevent the network from overloading. The congestion control algorithm presented in the following is the BBR-algorithm BBRv1 ("Bottleneck Bandwidth and Round-trip propagation time"-algorithm) developed at Google. It doesn't wait for packets being dropped but, to model congestion it uses changes in packet delays [14]. In case these inflight packets are taking longer to be acknowledged it assumes the connection is saturated and the packets have started to buffer. This algorithm is not loss-based and consists of actions to be taken upon receiving an acknowledgment at send-time.
Sending The number of packets that are send at time t depends on the product of the current Bottleneck Bandwidth and the current RTT multiplied by a constant value. [15] [14]BBR pretends to operate without creating packet loss or filling buffers. [16] It wants to achieve to figure out what its fair share of the bottleneck link is and to pace sending packets at exactly that rate. By permanently probing for bandwidth, it determines the sending rate.
Two main parameters allowing BBR to control what is done are: The pacing rate and the congestion window.
Pacing is used by BBR to control it´s sending behavior and is needed to match the packet-arrival rate to the bottleneck link's departure rate. The pacing mechanism permits servers to transmit data on high precision timers for millions of concurrent flows and enforce a maximum rate at which BBR schedules packets for transmission[16].
But, like in nearly every area, BBR is not in every scenario the perfect TCP congestion control algorithm. There exist some different ones to serve different scenarios. TCP Cubic[15] uses for example packet loss as a congestion signal and BBR does not. To build a congestion control algorithm one has to know the optimal operating point to do congestion control on the one hand and if the congestion signal should be packet loss on the other hand. The question, that need to be answered is in which scenarios BBR performs better than other congestion control algorithms and to determine when to employ BBR.

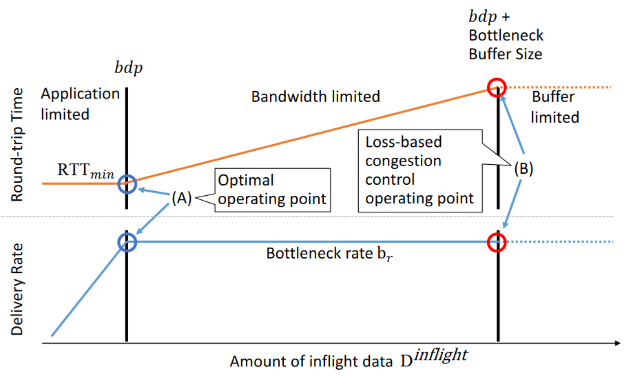

Fig. 1. Results of increasing inflight data on the connection's bandwidth and RTT

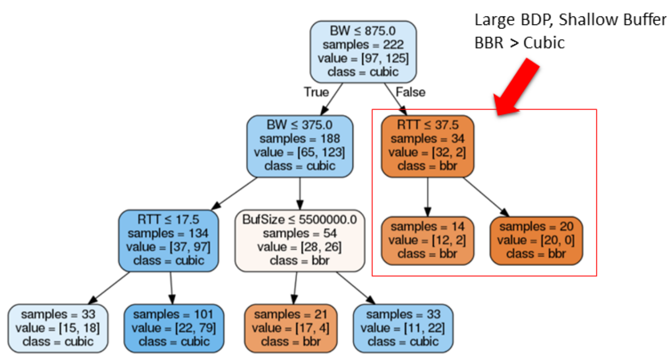
Large BDP, Shallow Buffer
BBR > Cubic
BW ≤ 875.0 samples = 222 value = [97, 125] class = cubic
True
False
BW ≤ 375.0 samples = 188 value = [65, 123] class = cubic
RTT ≤ 37.5 samples = 34 value = [32, 2] class = bbr
RTT ≤ 17.5 samples = 134 value = [37, 97] class = cubic
BufSize ≤ 5500000.0 samples = 54 value = [28, 26] class = bbr
samples = 14 value = [12, 2] class = bbr
samples = 20 value = [20, 0] class = bbr
samples = 33 value = [15, 18] class = cubic
samples = 101 value = [22, 79] class = cubic
samples = 21 value = [17, 4] class = bbr
samples = 33 value = [11, 22] class = cubic

Fig. 2. Decision tree generalizing the goodput values (5-200ms RTT, 10-1000Mbps bandwidth, 0.1-50Mb Buffer)

A. BBR vs. loss-based algorithms

In some cases loss-based congestion algorithms such as Cubic work worse than BBR, namely in the scenario where the bottleneck buffersize is much smaller than the bandwith-delay product (BDP). [17] BBR can achieve here 200% higher goodput than Cubic. But in the case, where the bottleneck buffersize is larger than the BDP, Cubic can achieve 30% higher goodput than BBR. BBR achieves therefore high good-put in shallow buffers but in shallow buffers BBR´s packetloss is higher than that of Cubic too, which is due to BBR´s configuration parameters maintaining 2*BDP of data in flight. BBR makes the number of inflight packets to be a multiple of the bandwidth-delay product[18]. To reduce packet losses one has to either decrease the 2* multiplier or to increase the bottleneck buffersize. [14] For example if bottleneck buffers are small, loss is misinterpreted as a signal of congestion by loss-based congestion, implicating low network utilization. If bottleneck buffers are large, they are kept full. This causes bufferbloat, which is the fact of having large buffers that take up data, being sent faster than the receiving side can consume it. This is illustrated in the following example: Everything works fast and fine if the data is coming into the network slower than the links can support. But once the data comes in faster than it can go out the data gets routed through the maze of buffers to hold it until it can be sent.

Figure 1 [18] shows us the following [13]: If less data than the BDP is inflight, there is no congestion and the RTT equals RTTmin. If the number of inflight data equals BDP[18], the bottleneck link is fully utilized, there is nearly no queing delay and, when the inflight data reaches the BDP, the delivery rate is on the maximum, which is shown with the optimal point of operation (A). To match the packet-arrival rate to the bottleneck link's departure rate, BBR paces every data packet [14].If one increases the amount of inflight data further, the packets arrive faster at the bottleneck than they can be forwarded filling the buffer[13]. The inflight-BDP excess creates queues and delays increasing lineary with the amount of inflight packets [14].Packets are dropped and RTT stopped when the excess exceeds the buffer capacity, namely after bdp + Bottleneck Buffer Size (operating point (B))[18]. Loss-based congestion control algoritms such as Cubic operate beyond this point.

A closer look to BBR versus other TCP congestion control algorithms show the following differences[17]:

1. In case of loss, BBR, on the contrary to Reno and Cubic, does not regard it as a congestion event. BBR uses the bandwidth and RTT to decide how many packets are sent..
2. BBR uses pacing rate, allowing to control the inter packet spacing, as one of the main control parameters. The congestion window, used to know how many data is inflight, is calculated differently by BBR and Cubic.
3. In response to congestion, loss-based algorithms reduce their sending rate whereas BBR does not use network con-gestion. BBR maintains 2*BDP worth of packets in flight. [13]BBR, on one side, avoids congestion by sending a limited number of packets where Reno and Cubic, on the other side, are going to send more and more packets in flight until firstly the bottleneck buffer is full and packet loss is detected. Problems could occur if the buffer isn´t shallow because bufferbloat could happen.

B. Analyzing who achieves when higher output

Analysing the TCP congestion control algorithms BBR and Cubic deeper helps to answer when BBR is useful compared to the loss-based algotithm Cubic or to answer if BBR is unfair to loss-based algortihms or even if BBR has higher output than Cubic in various scenarios. Results of such experiments, collecting the goodput values of BBR and Cubic under changing network settings in a LAN network, are shown in the decision tree in Figure 2 [17].

Blue nodes show cases where Cubic reaches higher goodput than BBR and orange nodes where BBR has higher goodput than Cubic. When BBR performs well is determined by the difference between the bottleneck buffer size and the BDP. This happens in case where BDP is large and the buffer size is shallow. On the opposite, Cubic works better and has higher goodput in case where the buffer size is deep and the BDP is shallow.

In the accomplished experiments, the following metric to

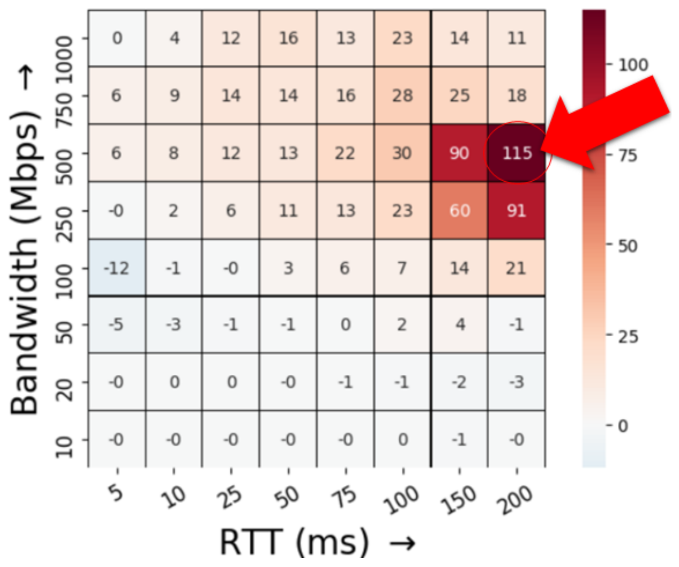

Fig. 3. Heatmap showing BBR´s GpGain with sallow buffer(100kb) in a LAN network

compare BBR's goodput gain to Cubic is used:
$GpGain = \frac{goodput(BBR)-goodput(Cubic)}{goodput(Cubic)} * 100$ [17]

Figure 3 represents a heatmap of the GpGain from the formula above for a shallow buffer of 100KB which is the result of an experiment with 640 iPerf3 transfers (1min) in a LAN network, under the following conditions:
1. Linux Traffic contol used to set network delay
2. Token Bucket filter used to set the buffer size and the network bandwidth.
It illustrates the GpGain under various RTT and network bandwidth values. For example, under 200ms RTT and 500Mbps bandwith, BBR performs better than Cubic (115 % more goodput). This can be explained by the fact that, in case of BBR, the congestion signal is not packet losses, such as it is for Cubic.

C. Considering BBR´s fairness to loss-based congestion algorithms

One question still left is if BBR is inequitable to loss-based algorithms such as Cubic. This is again shown in an experiment using a Mininet network, where BBR and Cubic TCP flows are created and send to the receiver at the same time over a 1Gbps link (RTT being 20ms & different buffer sizes). Figure 4 [17] illustrates this. The goodput changes for different buffer sizes. In case, where the buffer is small (20Kb), BBR can obtain more than 90 % of the total goodput. In case, where the buffer is large (10Mb), Cubic can obtain more than 80 % of the total bandwidth[13]. The discussion about the fact that BBR is unfair to Cubic depends therefore on the bottleneck buffer size.

D. BBR´s amount of goodput drops

Finally, after all the previous discussions about questions like if BBR is inequitable to loss-based congestion algorithms

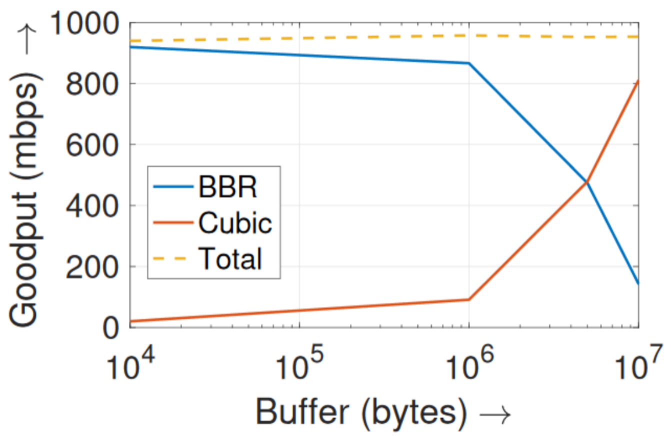

Fig. 4. BBR's and Cubic´s goodput is changing for varying buffer sizes, 1Gbps bandwidth & 20ms RTT

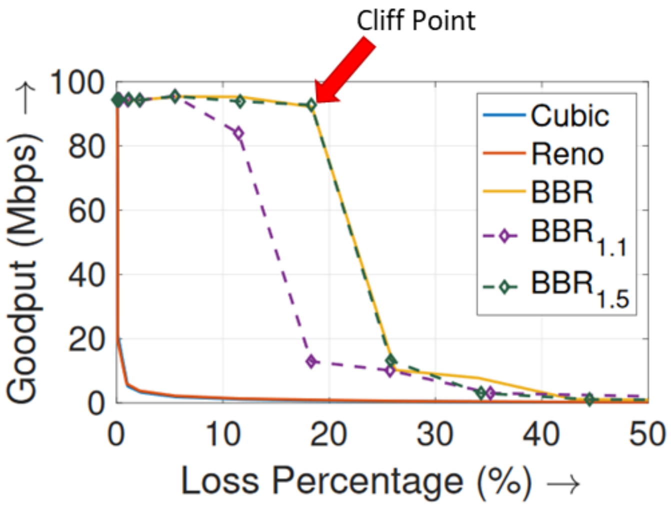

Fig. 5. BBR´s cliff point where goodput is shown dependant on the amount of loss (Mininet, 100Mbps bandwidth, 25ms RTT & 10Mb buffer)

such as Cubic or Reno, or BBR´s pacing technique, or when BBR realises more goodput than Cubic, it should have been noticed one last significant sign which is the following: BBR has a point, called cliff point, at which BBR loses amount of gooput which is shown in Figure 5 [17]. One can recognize that goodput depends on the loss rate and illustrates on the one hand that under loss-based algorithms like Cubic the goodput drops significally even if the loss rates are not important and on the other hand the decrease of BBR´s goodput. This goodput drop happens if the percentage of loss attains about 20%. The reason why this happens was explained in the experiment that, if the loss rate exceeds 20 %, BBR reacts to this indication. BBR should therefore update its guessed maximum bandwidth but, in reality, utilizes the unlimited average and standard bandwidth.

V. Conclusion

In general it can be said that pacing is not perfect. It does not perform better than unpaced congestion control algorithms in networks with larger buffers. But it can help to make TCP less bursty and thus usable in smaller buffered networks. In particular the edge pacing approach seems to yield very promising results but real world experiments will have to be conducted on this. Also it seems to be important for host paced TCP variants that all clients in the network use it because otherwise all hosts implementing it will lose bandwith. Considering the BBR congestion control algorithm part, there are some cases where loss-based TCP algorithms, such as Cubic, perform better, namely in case where BDP is shallow and the buffer size deep and some where BBR works better what´s the case if BDP is large and the buffer size shallow. A very similar scenario occurred looking at the bandwidth and the buffer size. In cases where the buffer is small BBR can obtain nearly 100% of the available bandwidth and in cases where the buffer is large Cubic can. Thus, it depends on the scenario whether BBR or Cubic achieve higher throughput.

References

[1] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Part III: routers with very small buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 35, 2005.

[2] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," ser. SIGCOMM '04. Association for Computing Machinery, 2004.

[3] E. Blanton and M. Allman, "TCP Congestion Control," accessed 2020-03-11. [Online]. Available: https://tools.ietf.org/html/rfc5681

[4] S. Floyd, A. Gurtov, Y. Nishida, and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," accessed 2020-04-28. [Online]. Available: https://tools.ietf.org/html/rfc6582

[5] L. Xu, A. Zimmermann, L. Eggert, I. Rhee, R. Scheffenegger, and S. Ha, "CUBIC for Fast Long-Distance Networks," accessed 2020-04-28. [Online]. Available: https://tools.ietf.org/html/rfc8312

[6] S. Villamizar, "High performance TCP in ANSNET," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 45–60, Oct. 1994.

[7] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Queue*, vol. 9, no. 11, Nov. 2011.

[8] V. Sivaraman, H. Elgindy, D. Moreland, and D. Ostry, "Packet pacing in small buffer optical packet switched networks," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 1066–1079, 2009.

[9] N. McKeown, G. Appenzeller, and I. Keslassy, "Sizing router buffers (redux)," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 5, pp. 69–74, Nov. 2019.

[10] H. H. Gharakheili, A. Vishwanath, and V. Sivaraman, "Edge versus host pacing of TCP traffic in small buffer networks," in *2013 IFIP Networking Conference*, May 2013, pp. 1–9.

[11] O. Alparslan, S. Arakawa, and M. Murata, "Performance of Paced and Non-Paced Transmission Control Algorithms in Small Buffered Networks," in *11th IEEE Symposium on Computers and Communications (ISCC'06)*, Jun. 2006.

[12] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Routers with Very Small Buffers," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, Apr. 2006, pp. 1–11.

[13] J. et Al., "Reproducible measurements of tcp bbr congestion control," *Computer Communications*, vol. 144, May 2019.

[14] C. S. G. S. H. Y. V. J. Neal Cardwell, Yuchung Cheng, "Bbr: Congestion-based congestion control measuring bottleneck bandwidth and round-trip propagation time," Dec. 2016.

[15] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a Deeper Understanding of TCP BBR Congestion Control," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, May 2018.

[16] Y. Cheng, N. Cardwell, V. Jacobson, and S. Yeganeh, "BBR Congestion Control," library Catalog: tools.ietf.org. [Online]. Available: https://tools.ietf.org/html/draft-cardwell-iccrg-bbr-congestion-control-00

[17] Y. Cao, A. Jain, K. Sharma, A. Balasubramanian, and A. Gandhi, "When to use and when not to use BBR: An empirical analysis and evaluation study," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. Association for Computing Machinery, 2019.

[18] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, Oct. 2017.