Impact of Buffer Size on a Congestion Control Algorithm Based on Model Predictive Control

Taran Lynn, Nathan Hanford, Dipak Ghosal
Department of Computer Science
University of California, Davis

ABSTRACT

For dedicated private WANs we propose a new congestion control algorithm based on Model Predictive Control (MPC) to produce flows with predictable rates and round trip times (RTTs). We model the bottleneck link as a queue, and based on a first order model relating the sending rate and the RTT we have designed and implemented (as a Linux kernel module) a MPC based congestion control algorithm that avoids the extreme window (which translates to rate) reduction that exists in current control algorithms when facing network congestion. Through simulation and experimental analyses, we have shown that our algorithm achieves the goals of a low standard deviation of RTT and pacing rate, even when the bottleneck link is fully utilized. In the case of multiple flows, we can assign different rates to each flow and as long as the sum of rates is less than the bottleneck rate, they can maintain their assigned pacing rate with low standard deviation. In this paper, we study the impact of network buffer size on the performance. Simulation analyses show that with a buffer size of at least 25% of the bandwidth delay product (BDP), the flows equally share the bottleneck link capacity and achieve full link utilization. Increasing the buffer size has an impact on the variance of the sending rate and RTT which is studied with respect to the number of flows.

ACM Reference Format:
Taran Lynn, Nathan Hanford, Dipak Ghosal. 2019. Impact of Buffer Size on a Congestion Control Algorithm Based on Model Predictive Control. In *Proceedings of Buffer Sizing Workshop (BS'19).* ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

The target domain of our work is an SDN-enabled softwarized private/dedicated WAN such as those interconnecting geo distributed data centers [10, 21] or an overlay network [5] inter-connecting distributed instrument facilities, HPC systems, storage, and researchers in the SuperFacility model [22]. In such networks, there is a need for sources (end-systems) to transfer data at an assigned rate predictably (i.e., with very low variance). This need may arise not only to meet deadlines (which are increasingly tight) [8, 11, 12, 23] but also to achieve high network utilization [10, 21].

One approach to achieve predictable data transfer is to use a rate based congestion control algorithm [4, 16]. These algorithms use a qDisc such as FQ/pacing and HTB [1] or a scalable traffic shaping function (potentially implemented on a programmable NIC) such as Carousel [19] to maintain a determined/assigned rate. The underlying control algorithm that determines the rate could be a delay-based algorithm [4, 16] that uses the RTT (time between when a packet is sent and the acknowledgement is received) to detect the congestion. These are opposed to loss-based algorithms that use loss to indicate congestion [7, 13]. There are also algorithms that use both delay and loss for control [14].

One of the earliest delay-based congestion control algorithms is TCP-Vegas [3]. More recently, BBR TCP [4] for WAN and TIMELY [16] for data centers have been proposed. In WANs, the key issue is that using a loss based approach pushes the control to operate at full buffer utilization, which can have negative consequences with large buffers. In BBR TCP, the control attempts to determine the rate that maximizes power, which is the ratio of throughput to the delay. This is referred to as Klienrock's optimal operating point as it achieves maximum throughput while minimizing the delay [4, 9]. In data center networks, studies show that changes in RTT are an accurate indication of congestion [16]. In our proposed algorithm we implement a delay-based congestion control algorithm.

We started with the goal to design and implement a congestion control algorithm from scratch following a formal control theoretic approach. We adopted model predictive control (MPC) and to the best of our knowledge, there are only limited prior studies on its use in the design of a congestion control algorithm [6]. MPC [2] is advantageous over other control strategies (like PID) when dealing with complex systems [18]. The main advantage is that unlike additive increase/multiplicative decrease controls, MPC can enable us to estimate our control as a continuous function. This in turn leads to smoother transmission rate and RTT. One caveat is that MPC methods require a system's reactions to correlate directly to the actions taken by the controller, and thereby also require that stochastic noise be relatively low. This makes it suited to the relatively quite dedicated WANs (our target domain) but not to the general Internet (which can be very noisy and unpredictable). Consequently, if successful, an MPC based rate control algorithm can lead to improved performance for long-lived, large data transfers (aka elephant flows) in dedicated WANs.

We have implemented the MPC-based congestion control algorithm in the Linux kernel. Through simulation and experimental results we have demonstrated that our algorithm achieves the goals of low standard deviation for pacing rates and RTT, while maximally utilizing the bottleneck rate. Our implementations allow per flow rates to be set. Our experimental results show that if this is used to cap flows so that their sum is less than the bottleneck rate, then we can achieve low standard deviation even among multiple flows that may have different RTTs.

In this paper we investigate the impact of buffer size on the bottleneck link utilization and the predictability of the proposed algorithm. Through simulation analysis we show that varying the bottleneck link buffer size has four major impacts. First, increasing the buffer size to a very large mutiple of the buffer-delay product (BDP) causes losses to increase. Second, decreasing the buffer size causes the rate and RTTs to become more stable. Third, decreasing

or increasing the buffer size to extremely small or large inhibits the algorithm from matching the bottleneck rate. Fourth, increasing the buffer size causes the algorithm to reach the bottleneck rate faster. These four aspects should be considered when setting a buffer size. Generally, a buffer size of 1/4 BDP may be preferable for long lived flows (i.e., those that last several minutes), while a larger buffer size (upwards of 4 times the BDP) is preferable for shorter lived flows.

2 MPC-BASED CONGESTION CONTROL

In MPC, the goal is to set a variable (called the **response**), which we can only indirectly manipulate using a value (called the **control**) that we control directly, such that it matches some desired value (called the **target value**)[15]. What makes MPC special is that it predicts the response for several time steps in the future, and tries to optimize the control for each of these time steps to match the response. This process is repeated at each time step, and is thus called the **receding horizon** (the idea being that the horizon of prediction recedes into the future as we gain more information). Below we describe the three core pieces of MPC as they apply to our control algorithm.

Model: The model or the physics of the system is used to predict and to optimize the control. We assume the network is a simple queue, and that our round trip time (RTT) increases as the queue becomes larger. From this we can establish several key parameters for the model. The first is the **propagation latency** l_P, which is the minimum RTT of the network, and corresponds to an empty queue. The second is the **bottleneck rate** r_B, which is the pacing rate that when exceeded leads to an increase in RTT, and corresponds to processing rate of the queue. The third is the **bottleneck latency** l_B, which is the maximum RTT for the network, and corresponds to a full queue. Taking these three parameters into account the model is as follows.

$$l_{\text{buf}}(n+1) = l_{\text{buf}}(n) + \frac{r(n) - r_B}{r_B}\Delta t(n) \tag{1}$$

$$l(n) = l_P + l_{\text{buf}}(n) \tag{2}$$

This is a discrete model indexed by n, where $t(n)$ is the time at index n, $\Delta t(n) = t(n+1) - t(n)$, and $r(n)$ and $l(n)$ are the pacing rate and RTT at time $t(n)$. $l_{\text{buf}}(n)$ is the portion of $l(n)$ that comes from network buffering (i.e., congestion). We must also impose the constraint that $0 \leq l_{\text{buf}}(n) \leq l_B - l_P$ for all n.

Prediction: We let $\hat{x}$ denote the predicted or estimated value of a variable x, and $\overline{x}$ denote its average. For l_P and l_B we simply take the minimum and maximum RTT with exponential back-off. The back-off is necessary to account for changing or erroneous minimum and maximum RTT values. Exponential back-off allows for quick recovery while maintaining algorithmic stability. Mathematically, the exponential back-off function f_{exp} is given by

$$f_{\text{exp}}(x, n) = x(n) + \frac{\bar{l} - x(n)}{t_d}\Delta t(n). \tag{3}$$

The above discrete function $f_{\text{exp}}(x, n)$ approximates the continuous exponential back-off function $\bar{l} + (x(t) - \bar{l})e^{-t/t_d}$ where t_d (a user set parameter) is the time it takes for the estimation to decay to 0.37 of its original value. Based on the above function, $\hat{l_P}(n+1)$ and $\hat{l_B}(n+1)$ are determined as follows

$$\hat{l_P}(n+1) = \min\left\{l(n), f_{\text{exp}}(\hat{l_P}, n)\right\} \tag{4}$$

$$\hat{l_B}(n+1) = \max\left\{l(n), f_{\text{exp}}(\hat{l_B}, n)\right\}. \tag{5}$$

To better estimate l_P and l_B, we also periodically decrease and increase the pacing rate to probe the RTT.

For r_B we use gradient descent to minimize the error between previous predictions for l and the measured value of l. It can be shown that

$$\hat{r_B}(n+1) = \hat{r_B}(n) - \eta\frac{\delta(\delta - \mu)}{\hat{r_B}(n)^3} \tag{6}$$

$$\delta = r(n)\Delta t(n) \tag{7}$$

$$\mu = (l(n) - l(n-1) + \Delta t(n))\hat{r_B}(n) \tag{8}$$

where η is the learning rate, and δ and μ are terms that come out of the derivation. We then plug these equations into Eq.(1) and Eq.(2) to get the predicted latency $\hat{l}(n+1)$.

Optimization: To optimize the pacing rate we first establish a target latency l_t (usually set by the user somewhere between l_P and l_B). We then set the rate to minimize the difference between the predicted latency $\hat{l}$ and target latency l_t, the predicted variance in l, and the variance of the pacing rate. These three objectives are given the weights $0 < 1 - c_1 - c_2 < 1$, $0 \leq c_1 < 1$, and $0 < c_2 < 1$, respectively. These weights are set by the user. The resultant formula is

$$r(n+1) = \frac{c_3 l_P^2 r(n) - \Delta t(n)\hat{r_B}(n)\Lambda}{c_3 l_P^2 + \Delta t(n)^2(\alpha c_2 + c_1)} \tag{9}$$

$$\Lambda = (\alpha c_2 + c_1)\left(l(n) - \Delta t(n)\right) - c_1 l_t - \alpha c_2 \bar{l}(n) \tag{10}$$

where α is the weight used for the exponentially weighted moving average of l, i.e., we compute $\bar{l}(n+1) = (1-\alpha)\bar{l}(n) + \alpha l(n)$. We set α to $\frac{1}{8}$. Λ is a common value that comes out when deriving the formula, and represents the RTT part of the optimization.

3 SUMMARY OF EXPERIMENTAL RESULTS

We have implemented the MPC based congestion control algorithm as a Linux Kernel module. The algorithm only requires changes at the sender side. Using the implementation we ran tests over a 10 Gbps link to ESnet's test servers [5]. For these tests the 10 Gbps link was the primary bottleneck. We used the pschedular tool, which is part of the perfSonar suite [20]. One important feature of pschedular is that it ensures that we are not in contention for the server before starting an iPerf3 [17] session to measure network performance. Target servers were selected to give a broad range of RTTs; it included servers in Sacramento, Denver, and New York. Overall, the results, some of which are discussed below, were similar to those observed from the simulation analysis.

For a single flow between Davis, CA and Denver, CO we achieved rates and RTTs that were stable and did not show a saw tooth pattern as found in other AIMD-based congestion control algorithms. Results using multiple flows to Denver, CO showed stable rates and RTTs as well, and the flows would equally share the bandwidth. This equal sharing occurs when flows have the same RTT. When

testing different RTTs we used New York, NY as a destination in addition to Denver, CO. The results showed that Denver would dominate the bandwidth usage, showing that the algorithm favors lower RTTs, at least for the parameters we set. Preliminary experimental results with detailed explanations are available in [15].

Overall, the experimental results demonstrated four things. First, for a single flow we can achieve near bottleneck rates while also maintaining a smooth RTT. Second, when we specify rate caps we can maintain the rate with low standard deviation and achieve a smooth RTT. Third, with multiple uncapped flows with the same RTT, the flows approximately equally share the bottleneck capacity. Fourth, with multiple uncapped flows with different RTTs, one flow dominates.

4 IMPACT OF BUFFER SIZE

4.1 Simulation Methodology

To study the impact of buffer size on the performance of the algorithm we have used a discrete-event simulation model. The network is modeled with two buffers; one to model the bottleneck link, and the other for the returning ACKs. The simulator allows separate client processes representing end systems to be initiated each running their own instance of the MPC algorithm, which is near identical to the Linux kernel implementation. Three types of events, namely, enqueue, dequeue, and ACK, are modeled. Enqueue represents a client sending a packet and enqueueing it on the bottleneck link's buffer. The enqueue events are specific to each client; the event being scheduled at time intervals determined by the rate set by the MPC algorithm and the size of each packet. If an enqueue occurs and the bottleneck link buffer is full, a loss is recorded. Dequeue corresponds to the bottleneck link processing and transmitting the next packet in its queue. The dequeue events are scheduled at time intervals determined by the rate set by the bottleneck rate and the size of each packet. ACK is scheduled to occur after one RTT propagation delay from dequeue time, with a small added noise sampled from a bounded (negative) exponential distribution with mean 1% and a max of 10% of the base RTT. When an ACK is received the RTT is calculated and passed to the MPC algorithm, which updates it's sending rate. In our setup we used a shared link of 40 Gbps with a base RTT of 25 ms.

4.2 Results

To evaluate the impact of buffer size we simulated different numbers of flows that shared a 40 Gbps bottleneck link. We changed the bottleneck link's buffer size to varying percentages of the bandwidth-delay product (BDP). Of the approximately 100,000,000 data points we sub-sampled 1,000,000. Unless stated otherwise, all plots correspond to a system with two simultaneous flows running. For the box plots in this paper no points were considered outliers, and the whiskers thus represent the full range of data. This was done because the distribution is heavily tailed due to probing, and thus using the 1.5 IQR method or a similar outlier detection method would exclude a large portion of the data.

Figure 1 shows the effect of changing the buffer size on the achieved rate. For buffer sizes less than 1/4 the BDP or greater than 2 times the BDP, the throughput rate starts to be limited to less than the bottleneck rate. For small buffer sizes, since the RTT range is more limited (see Figure 2), the algorithm is limited in the control actions it can take and is thus more cautious. For large buffers this is caused by a high RTT variance (see Figure 2), which causes the algorithm to overreact, leading to high losses resulting in the algorithm constantly backing off. As buffer sizes increase beyond 1/4 the BDP, the variation in the rate increases. This is because as the range in RTT increases, the algorithm must take greater control action to meet the target RTT (which is halfway between the minimum and maximum RTT). The large range in rate is caused by the algorithms probing cycle. The minimum whiskers in the box plot correspond to the algorithm probing for the minimum RTT, where it decreases the rate to empty the link buffer, which in turn decreases the RTT. The maximum whiskers in the box plot correspond to the algorithm probing for the maximum RTT, where it increases the rate to fill the link buffer, which in turn increases the RTT. One may notice that when probing for a maximum RTT the pacing rate goes beyond the bottleneck rate. This is acceptable because it only occurs for a short time span, and the buffer is able to absorb the excess packets. We can see this in Figure 5, where the spikes correspond to probing. The probing also immediately stops when a loss is detected. The figure also demonstrates one advantage the MPC based controller has over traditional AIMD based controllers. Because the control acts as a continuous function, when the algorithm is not probing we can produce a low variance rate for moderate ($\leq$ 1 BDP) buffer sizes that tracks very close to the bottleneck rate. This is opposed to AIMD controllers, whose pacing rate (a function of the window size) exhibits a saw tooth pattern, leading to high variation in the rate.

Figure 2 shows the effect of changing the buffer size on the RTT. We can see that as buffer size increases so does the range of RTTs. This corresponds to the core idea of our algorithm that fuller buffers result in larger RTTs. The explanation for this is that packets must wait for the ones in front of them to be transmitted first before they can be transmitted, and thus the packets' RTTs increase. Note that since the RTT set by our control algorithm is a function of the RTT range, we will find that the median RTT increases with buffer size as well. The whiskers again correspond to probing by the algorithm. The lower end of the boxes end abruptly at 25 ms because that is the propagation delay (i.e. the minimum RTT). As with the rate, at a reasonable buffer size the algorithm can maintain a low variance RTT and avoid the saw tooth pattern of AIMD protocols.

Contrary to what is expected, a larger buffer size has little effect on losses until it reaches 8 BDP, at which point the number of losses starts to increase. For sizes less than 8 BDP, the number of packets dropped is on the order of around 10 packets per 100,000,000. In contrast, at 16 BDP we lose 176 packets. This increase is due to the large range in RTT and the resulting instability. In order to meet the target RTT the algorithm will set the pacing rate high in order to fill the buffer. However, because the RTT can increase to a high value, the feedback signal is significantly delayed. This causes the algorithm to overshoot and incur high losses. Upon seeing these losses the algorithm will cut the rate and repeat the process. This leads to a high number of losses and a low median pacing rate.

A summary of how the rate, RTT, and losses change with respect to the number of flow and buffer size is shown in Table 1. The pacing rate is largely affected by buffer size, with the number of flows having a greater impact with very small or large buffers. For

small buffers, more flows leads to more competition, which pushes the combined rate up even when an individual flow rate is low. Conversely, for large buffers this effect magnifies the algorithmic instability observer at these sizes, leading to lower median pacing rates. For the RTT, low buffer sizes generally result in the RTT being close to its base RTT for any number of flows. At higher buffer sizes, the RTT for different numbers of flows diverge, with more flows leading to a higher RTT. This is because packets from each individual flow must wait for packets from every other flow to be processed before it can be processed, leading to a higher average RTT. Losses generally increase with the number of flows, which is due to congestion caused by increased competition. At very high buffer sizes, losses also increase due to algorithmic instability.

Results seem to indicate that a buffer size of 1/4 BDP is ideal. This buffer size leads to a minimal increase in RTT, achieves maximum throughput, has a low median RTT, and has the lowest variance in both throughput and RTT. However, one thing these graphs do not show is how long the flows take to reach the bottleneck rate. If we look at Figure 4 and 5 we can see how changing the buffer size affects how long the combined flows take to reach the bottleneck rate. In general increasing the buffer size causes the algorithm to reach the bottleneck rate faster. This is because, as mentioned earlier, it does not need to be as cautious to avoid losses. A different effect can be seen when flows start at different times. Figure 6 and Table 2 show what happens when one flow starts 10 seconds after another one. The flows take a significant time to stabilize, and this time increases when the buffer size increases. This increase is in part due to the higher RTT range making it more difficult for the flows to separate the signals from their probes. It is important to note that the flows will also stabilize faster if they probe more frequently, although this increases the variance of the pacing rate. A larger buffer size (upwards of 4 BDP) may be preferable because it reaches the bottleneck rate faster. In general smaller buffers (1/4 BDP) are preferable for long lived flows, where the long startup time is offset by the increased stability. On the

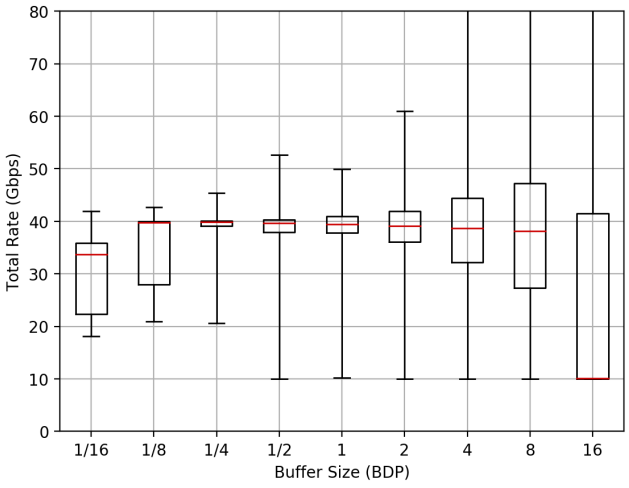


Figure 1: Combined rates for 2 flows versus buffer size in terms of BDP. Rates are most stable when the buffer size is at 1/4 the BDP. Decreasing the buffer size far below or above the BDP limits the rate to below the bottleneck rate. The large whisker range is due to probing for l_P and l_B, which correspond to the minimum and maximum rates, respectively. All boxes have a lower limit of 10 Gbps because each flow was set for a minimum pacing rate of 5 Gbps (two flows thus always use 10 Gbps). The whiskers for buffer sizes of 4, 8, and 16 times the BDP have been cutoff to better present the box plots, and they each extend up to 120 Gbps.

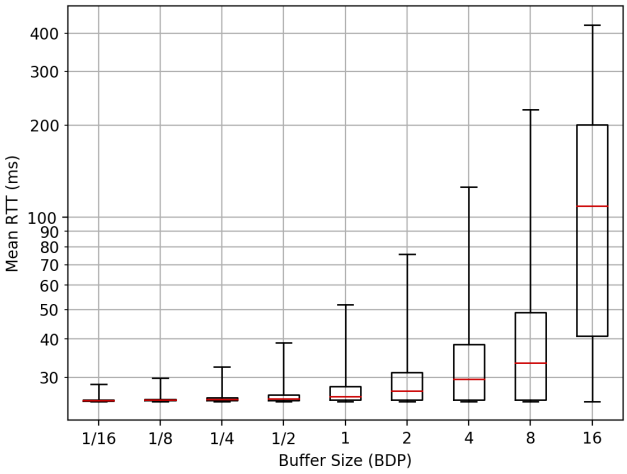


Figure 2: The RTT as a mean among the flows compared to buffer size in terms of BDP. The median and range in RTT increase as buffer size increases. The large whiskers correspond to probes for l_P and l_B. All boxes have a lower limit of 25ms because that is the minimum RTT l_P. Note that a log scale is used to accommodate the large variation in RTTs.

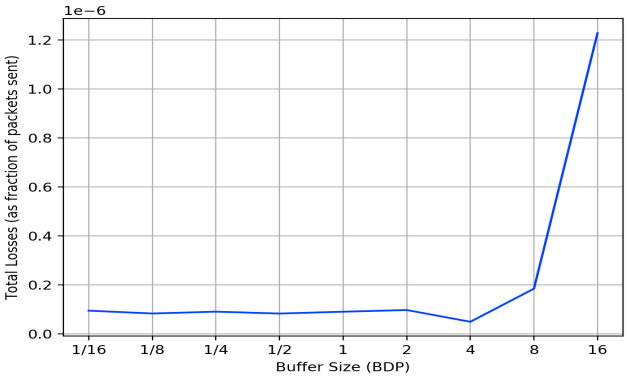


Figure 3: Total losses among all flows as a function of the buffer size in terms of BDP. Losses as a fraction of packets sent remain consistently around 1×10^{-7} until the buffer size reaches 8 BDP, at which point the number of losses increase due to algorithmic instability caused by a large range in RTT.

Flows	Buffer Size (BDP)								
	1/16	1/8	1/4	1/2	1	2	4	8	16
Median Combined Rate (Gbps)									
1	24.5	34.4	39.8	39.8	39.6	39.4	39.2	39.2	38.4
2	33.8	39.8	39.8	39.6	39.4	39.2	38.7	38.2	10.1
4	35.8	39.7	39.9	39.7	39.6	39.0	38.0	36.5	10.0
8	39.1	39.7	39.8	39.9	39.8	39.7	38.2	26.7	10.2
Median RTT (ms)									
1	25.2	25.2	25.3	25.5	25.9	26.8	27.5	27.9	28.9
2	25.2	25.3	25.4	25.5	26.0	27.1	29.6	33.4	109.1
4	25.2	25.3	25.4	25.5	25.7	27.1	32.2	41.5	165.8
8	25.3	25.3	25.4	25.4	25.4	25.5	33.5	59.5	175.4
Losses (as Fraction of Packets Sent)									
1	4.42e-8	3.42e-8	3.72e-8	3.51e-8	3.50e-8	2.77e-8	7.04e-9	2.84e-8	1.95e-7
2	9.54e-8	8.38e-8	9.13e-8	8.36e-8	9.11e-8	9.80e-8	5.01e-8	1.85e-7	1.23e-6
4	1.83e-7	1.55e-7	1.65e-7	1.30e-7	9.58e-8	1.48e-7	3.59e-7	5.05e-7	2.23e-6
8	4.09e-7	3.42e-7	2.40e-7	2.44e-7	2.31e-7	2.32e-7	4.88e-7	2.13e-6	3.47e-6

Table 1: The pacing rate and RTT are mainly affected by buffer size. For very small buffer sizes, increasing the number of flows increases the median combined rate. For very large buffer sizes, increasing the number of flows decreases the median combined rate, and increases the median RTT. Finally, an increased number of flows universally leads to greater losses, with higher losses observed at very large buffer sizes.

other hand, larger buffers allow the algorithm to react quickly at the expense of stability, which is preferable for short lived flows.

5 CONCLUSIONS

We have designed and implemented a rate-based congestion control algorithm based on the principles of MPC for dedicated WANs. The control algorithm uses a simple one time step queuing model of the bottleneck link to control the pacing rate by using RTT as a feedback mechanism. The formal MPC-based approach allows us to achieve a stable and continuous pacing rate, and avoids the saw tooth pattern that is typical in other AIMD-based algorithms. In this paper we presented simulation results on how our algorithm reacts to the bottleneck link buffer size. Results show the tradeoffs between buffer size, the predictability (variance) of the achieved rate, bottleneck link utilization, the response time, and convergence of bandwidth sharing. Generally, a buffer size of 1/4 BDP may be preferable for long lived flows (i.e. those that last several minutes), while a buffer size matching or larger than the BDP may be preferable for shorter lived flows. More experimental results are need to to further evaluate the benefits of our approach. Future work will also focus on developing methods to allow the algorithm to react faster and improve fairness while maintaining a low variability in the rate . Nevertheless, our formal MPC-based framework can serve as a benchmark for existing and new congestion control algorithms for dedicated WANs and the impact of network buffer sizes on their performance.

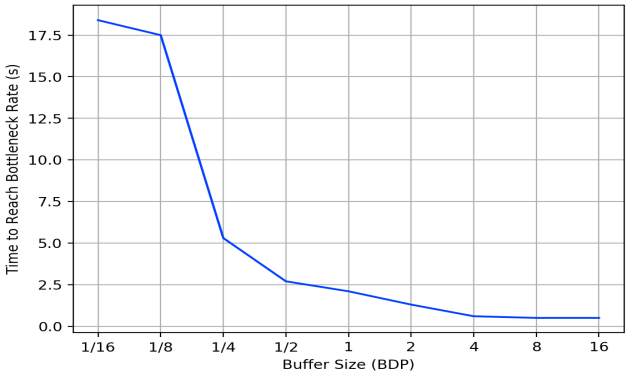


Figure 4: As buffer size increases,the time it takes for the algorithm to reach the bottleneck rate decreases. This is because the higher variability in the rate and RTT correlates to a lower response time.

ACKNOWLEDGEMENTS

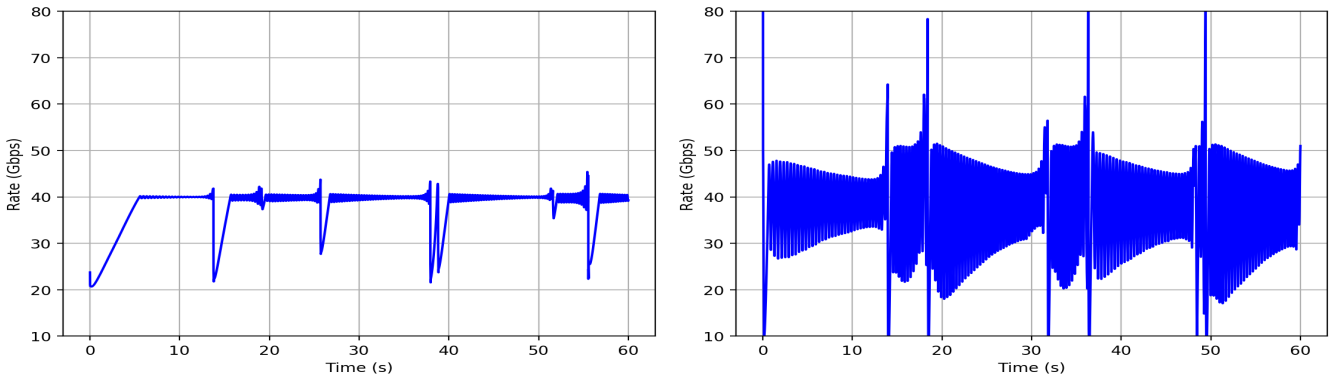
This research was funded by National Science Foundation (NSF) grant CNS-1528087 and the associated REU grant.

REFERENCES

[1] Doru Gabriel Balan and Dan Alin Potorac. 2009. Linux HTB queuing discipline implementations. In *2009 First International Conference on Networked Digital*

Buffer Size (BDP)	1/16	1/8	1/4	1/2	1	2
Time to Converge (s)	1	1	19	26	26	42

Table 2: The time for two flows, one starting 10 seconds after the other, to converge to a fair share of the bandwidth is influenced by the buffer size. Time for flows to converge is considered to the point when their Jain's fairness index reaches and maintains a value of at least 0.9. The table shows that flows take more time to converge with larger buffer sizes. The omitted buffer sizes never converged.



(a) 1/4 BDP

(b) 4 BDP

Figure 5: Plot of combined rate for multiple flows vs time. A buffer size of 1/4 the BDP takes about 5.5s to reach the bottleneck rate, while a size of 4 times the BDP takes about a second. Generally, higher buffer sizes allow the algorithm to reach the bottleneck rate faster.

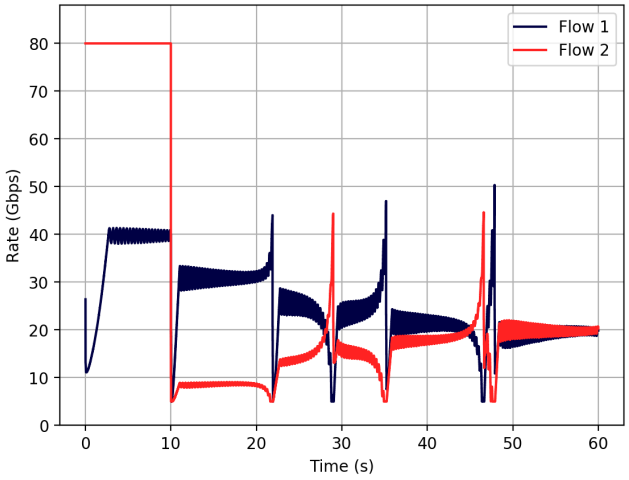


Figure 6: Flows take longer to converge to a fair share of the bandwidth when starting at different times than when they start at the same time. Here flow 2 starts 10 seconds after flow 1 with a buffer size of 1 BDP.

Technologies. IEEE, 122–126.
[2] Carlos Bordons and Eduardo Camacho. 2007. *Model Predictive Control* (2 ed.). Springer-Verlag London, London, England.
[3] Lawrence S. Brakmo and Larry L. Peterson. 1995. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE Journal on selected Areas in communications* 13, 8 (1995), 1465–1480.
[4] Neal Cardwell et al. 2016. BBR: Congestion Based Congestion Control. *acmqueue* 14 (2016), 20–53. Issue 5.
[5] ESnet. [n. d.]. Energy Sciences network (ESnet). ([n. d.]). http://www.es.net/.
[6] David Fridovich-Keil, Nathan Hanford, Margaret P Chapman, Claire J Tomlin, Matthew K Farrens, and Dipak Ghosal. 2017. A model predictive control approach to flow pacing for TCP. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 988–994.
[7] Shangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP variant. *Operating Systems Review* 42 (2008), 64–74. Issue 5.
[8] Željko Ivezić, Steven M Kahn, J Anthony Tyson, Bob Abel, Emily Acosta, Robyn Allsman, David Alonso, Yusra AlSayyad, Scott F Anderson, John Andrew, et al. 2019. LSST: from science drivers to reference design and anticipated data products. *The Astrophysical Journal* 873, 2 (2019), 111.
[9] Benedikt Jaeger, Dominik Scholz, Daniel Raumer, Fabien Geyer, and Georg Carle. 2019. Reproducible measurements of TCP BBR congestion control. *Computer Communications* 144 (2019), 31–43.
[10] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 3–14.
[11] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. 2014. Calendaring for wide area networks. In *ACM SIGCOMM computer communication review*, Vol. 44. ACM, 515–526.
[12] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, et al. 2015. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 1–14.
[13] Douglas Leith and Robert Shorten. 2004. H-TCP: TCP for high-speed and long-distance networks. *Proceedings of PFLDnet* 2004 (2004).
[14] Shao Liu, Tamer Başar, and Ravi Srikant. 2008. TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks. *Performance Evaluation* 65, 6-7 (2008), 417–440.
[15] Taran Lynn, Nathan Hanford, and Dipak Ghosal. 2020. A Model Predictive Control Based Network Traffic Control for Predictable Data Transfer. In *Under review for publication in 2020 American Control Conference*.
[16] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 537–550.
[17] Mathijs Mortimer. 2018. Iperf3 documentation. (2018).
[18] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. 2013. PIE: A lightweight control scheme to address the bufferbloat problem. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*. IEEE, 148–155.
[19] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Carlo Contavalli, Amin Vahdat, et al. 2017. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 404–417.
[20] Brian Tierney, Joe Metzger, Jeff Boote, Eric Boyd, Aaron Brown, Rich Carlson, Matt Zekauskas, Jason Zurawski, Martin Swany, and Maxim Grigoriev. 2009. perfsonar: Instantiating a global network measurement framework. *SOSP Wksp. Real Overlays and Distrib. Sys* (2009).
[21] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. 2011. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 50–61.
[22] Kathy Yelick. 2017. A Superfacility Model for Science. https://people.eecs.berkeley.edu/~yelick/talks/data/Superfacility-TechX17.pdf. (2017).
[23] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. 2017. Guaranteeing deadlines for inter-data center transfers. *IEEE/ACM Transactions on Networking (TON)* 25, 1 (2017), 579–595.