# Toward Fair and Efficient Congestion Control: Machine Learning Aided Congestion Control (MLACC)


## ABSTRACT

Emerging inter-datacenter applications require massive loads of data transfer which makes them sensitive to packet drop, high latency, and fair resource sharing. However, current congestion control (CC) protocols do not guarantee the optimal outcome of these metrics. In this paper, we introduce a new CC technique, Machine Learning Aided Congestion Control (MLACC), that combines heuristics and machine learning (ML) to improve these three network metrics. The proposed technique achieves high level of fairness, minimum latency, and minimum drop rate. ML is utilized to estimate the ratio of the available bandwidth of the bottleneck link while the heuristic uses this ratio to enable end-points to cooperatively limit the shared bottleneck link utilization under a predefined threshold in order to minimize latency and drop rate. The key to achieving the desired fairness is using the gradient of the link utilization to control the sending rate. We compared MLACC to BBR (which is at least on par with the state-of-the-art ML-based techniques) as a base case, in different network settings. The results show that MLACC can achieve lower and more stable end-to-end latency. It also significantly reduces packet drop rates while attaining a higher fairness level. The only cost for these advantages is a small throughput reduction of less than 3.5%.



## KEYWORDS

Congestion Control, Machine Learning, Gradient, LSTM, Rate-based Congestion Control


## 1 INTRODUCTION

With the increase in accessibility and popularity of the internet, the demand for the quality and efficiency of internet services grows as well. Technological advancements have led to the widespread execution of parallel computing applications within and between datacenters. These applications such as distributed training [17], SPark [21], and MapReduce [6] work by continuously alternating between communication and computation phases. During the communication phase, the data required to execute parallel tasks are transferred over the network to other machines or other datacenters. The computation phase can only start after completing the transfer of all the required data. Applications using such parallel computation paradigms are oversensitive to latency and packet drop as well as fairness among different data transfer connections. For instance, if a data transfer connection for a sub-task experiences high packet loss rate, the whole task will be blocked waiting for the sub-task to be completed. The same situation will also occur if the end-to-end latency for a sub-task is significantly longer than others. As a result, these communication metrics can significantly increase the task completion times, resulting in an overall degradation of the parallel application performance. In addition to these new inter-datacenter services, traditional applications such as hypertext transfer protocol (HTTP), file transfer protocol (FTP), and real-time communications (RTC) are also significantly affected by these metrics, and their performance can be substantially improved by optimizing these factors.

Congestion control (CC) techniques aim to improve network performance by preventing or mitigating congestion if it occurs. CC is an important component in data and network communication that determines the three major parameters: latency, drop rate, and fairness, in addition to throughput. However, none of the current TCP congestion control algorithms can satisfy all these requirements. For instance, the state-of-the-industry (SOTI), bottleneck bandwidth and round-trip time (BBR) congestion control algorithm [12], tries to minimize the average latency and achieve maximum link utilization. However, several studies show that while BBR works quite well for a single flow at a bottleneck link, it suffers several limitations in different scenarios [4, 10]. Through extensive experimental evaluation, the authors in [10] concluded that BBR leads to a sustained overload of the bottleneck which results in the increase of in-flight messages and longer queuing times at the bottleneck buffer, subsequently leading to increased queuing delays in case of large buffers as well as a massive amount of packet loss. This experimental evaluation of BBR also showed that it has no mechanism to converge to a fair share of bottleneck link among different BBR flows. Moreover, the authors in [23] demonstrated that BBR incurs a round trip time (RTT) fairness problem where a longer RTT flow would dominate a competing flow with shorter RTT.

Motivated by the aforementioned emerging technologies and the limitations of the SOTI, we propose a novel congestion control technique, MLACC, that combines heuristic with machine learning (ML) to take advantage of the successes in the ML field while avoiding the problems like low generalization that ML can bring. The goals of this CC technique are: **1)** to minimize the queuing at the bottleneck buffers (consequently minimizing the queuing delay and packet drop) and **2)** at the same time achieving a fair share of the bottleneck's resources. The key to achieve these goals is twofold.

The first objective is achieved by allowing the rate controller of the proposed CC algorithm to estimate the available bandwidth ratio ($a_t$) of the bottleneck using a ML model. Then, the heuristic controls the sending rate based on the bottleneck link utilization $u_t = 1 - a_t$ obtained from the estimation. The bottleneck link utilization can be considered as a common signal from the network to different senders sharing the same bottleneck. Based on this signal, those senders can coordinate to change their rates in such a way that avoids overloading the bottleneck link. In some scenarios, such as in-datacenter networking, the ML model can be replaced by the in-band network telemetry (INT) [18] if all nodes are INT capable. However, that is not guaranteed over the Internet. In this paper we focus on using ML since it can work in all scenarios and does not require any network hardware changes.

The second objective is fulfilled by using the gradient of the available bandwidth ratio ($a_t$) with respect to the sending rate to adjust the rate and the congestion window (CW). Using this gradient enables different senders sharing the same bottleneck to converge

and obtain a fair share of available bandwidth of the bottleneck link. The contributions of this paper are:

- We propose a new congestion control technique that aims to minimize the queuing delay and drop rates at the bottleneck link and increase the fairness among different connections sharing the same bottleneck.
- We developed a long-short-term-memory deep neural network model and trained it on simulated data to estimate the ratio of the available bandwidth at the bottleneck link. This estimation is used by the heuristic to adjust the CW.
- The proposed algorithm is developed in NS3 [9] platform and evaluated against TCP-BBR in different scenarios. We selected BBR as a benchmark since it is on par with the state-of-the-art (SOTA) ML-based techniques [19]

The rest of the paper is organized as follows. Section 2 provides an overview of the different congestion control techniques and outlines their advantages and limitations. Then, in Section 3, we explain the proposed framework including the heuristic and the different ways to obtain the bottleneck utilization. It also provides the details of the ML model as well as the prediction accuracy results. The comparison with BBR is introduced in Section 4 along with discussions of our findings, followed by the final conclusion and future extentsions in Section 5.

## 2 RELATED WORK

End-to-end CC systems are typically classified into loss-based, delay-based, and hybrid. Traditional loss-based approaches such as Reno[11] and CUBIC [15] use losses as congestion signals and are based on additive-increase/multiplicative-decrease. These approaches aim to achieve high throughput by constantly filling the link buffer, in which buffer bloat can occur if bottleneck buffers are large. It can also result in lower throughput in case of smaller buffers as the increase in loss may be misinterpreted as congestion. This causes longer delay, making it not suitable or ideal for delay-sensitive applications.

Delay-based CC methods such as TCP BBR [5] and TCP Vegas [3] utilize measured transmission delays to regulate the sending rate. They are better suited for high-speed and dynamic networks, like wireless networks, since they are not influenced by random packet loss. However, accurately measuring the transmission delay still poses a significant challenge [14] along with the other problems mentioned in Section 1 regarding fairness, link overutilization, and significant packet loss especially with BBR [16].

Other heuristics, outside the hard-wired mappings in traditional heuristics, address specific network scenarios based on the environment's characteristics. For instance, Sprout [20] concentrated on cellular networks, whereas label distribution protocol (LDP) [22] and datacenter TCP (DCTCP) [1] targeted datacentre networks. Such traditional CC techniques may work well in certain scenarios, but they cannot guarantee high performance in all network scenarios and changing traffic patterns can also affect their performance. Therefore, an intelligent CC approach such as ML algorithms needs to be considered.

The advancements in ML and artificial intelligence (AI) inspired research community to utilize these technologies in congestion control such as [2, 8, 13, 24]. The SOTA ML-based CC is Aurora, which depends completely on a deep-reinforcement learning (DRL) model. Statistics about latency and the ratio of packets sent to those acknowledged are used by the agent in Aurora as states and actions, respectively. It has been demonstrated that Aurora outperforms one of the most popular TCP variants, CUBIC, while being on par with BBR. Another example for ML-based CC is presented in [2], which is a hybrid bandwidth predictor for RTC. It utilizes an initial heuristic-based approach then switches to a full RL-based model that attempts to learn from the network statistics such as receiving rate, packet loss, and packet delay. However it is only tested for RTC since it uses the statistics generated by the RTC receiver.

Compared to SOTA CC techniques such as [2, 13], our proposed technique does not depend completely on ML to avoid the ML problems such as the lack of generalization of the offline trained models as well as exploration problems of the online DRL-based models. Instead, it integrates heuristics with ML, where the ML model assists the heuristic by estimating the available ratio of the bottleneck bandwidth. The heuristic, which is designed based on network fundamentals, can be generalized to wide range of network settings.

## 3 THE PROPOSED FRAMEWORK

This section explains the proposed framework and its components. It starts by discussing the idea behind the algorithm then detailing the operation of different components.

### 3.1 Background

The first question we asked when addressing this problem was, "why would smart CC algorithms like BBR and CUBIC cause packet drop and experience a queuing delay in some scenarios". The main cause of these issues is that they try to achieve high bandwidth utilization. For instance, BBR in its bandwidth probing phase, aggressively increases its rate to achieve the highest possible rate, which can overwhelm the bottleneck, resulting in longer queuing delay and possibly packet loss [10].

To solve this problem, MLACC defines a bandwidth utilization threshold $u_{thr}$ and tries to maintain the bottleneck utilization around it with a tolerance factor $\gamma$. The advantage of this method is that packet drop and queuing delay will be minimized since both are exponentially increasing with the link utilization. It periodically (every $50ms$ in our setting) estimates the average utilization $u_t$ of the bottleneck over the last interval using the ML model. If $u_t > u_{thr} + \gamma$, the link is overutilized and the sender tries to decrease its rate, and vice versa for the underutilization state. If $(u_{thr} + \gamma > u_t > u_{thr} - \gamma)$, the link utilization is in acceptable range. In this case, the sender tries to carefully increase its rate to respond quickly to any change in network conditions. Figure 1 compares the operating points of BBR to the proposed model, which shows how we sacrifice a small ratio of the throughput to achieve our CC design objectives.

There are two challenges facing this idea. The first being how the sender, which resides in the end-node, can obtain the bottleneck link utilization. The second challenge is that even if the sender can accurately obtain the bottleneck link utilization for its path, it does not know the varying amount of traffic flows using the same bottleneck.

In our approach, the first challenge is solved by using a ML model to estimate the link utilization from the connection history.

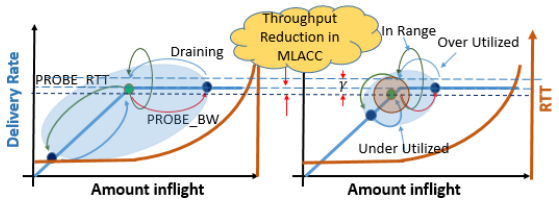



**Figure 1: The Operation of MLACC vs. BBR.**

In particular, we developed a long-short-term memory (LSTM) deep neural network (DNN) model that uses the connection history statistics such as latency and packet loss rate to estimate the available ratio of the bandwidth ($a_t$). This ratio is the complement of the link utilization. Details of the model will be explained in Section 3.4. To solve the second challenge, we developed a heuristic that uses the estimated available ratio of the bottleneck bandwidth to adjust the rate and congestion window based on the gradient of $a_t$ with respect to the sending rate.

## 3.2 How to increase the rate without overwhelming the bottleneck?

The main idea behind using the available bandwidth ratio $a_t$ to update the rate is illustrated in Algorithm 1. In this algorithm, in order to avoid overloading the bottleneck, the sender assumes the worst case. To understand the worst case, we can imagine a sender receiving $a_t$ for the bottleneck in the current time step. There are two extreme cases: 1) there are multiple senders (i.e., $n$) using this bottleneck, or 2) this sender is the only one using the bottleneck. In the first case, the sender contributes to the link load, on average, by $\frac{1}{n} \times L_t$ (where $L_t$ is the current link load). If it increases its rate by factor $f$, it will increase the load on the bottleneck by the ratio of $\frac{f}{n}$, in average. It is clear that a smaller $n$ means higher impact of the sender on the link load. Therefore, from the sender's perspective, $n = 1$ (i.e., only itself is using the bottleneck) is the worst case since it will increase the load on the link by the same factor $f$. In order to avoid overloading the bottleneck link, the sender updates the rate based on this worst case assumption. Based on this assumption, the available BW can be calculated as $bw = \frac{r_t}{1-a_t} = \frac{r_t}{u_t}$, where $r_t$ is the current sending rate. As Shown in Algorithm 1, the sender checks if the current link utilization is far from the target utilization with a predefined distance $\gamma = 0.05$. It will calculate the available bandwidth (based on the worst case assumption) and increase the rate by a factor $f_t$ of the available bandwidth, i.e., $r_{t+1} = r_t + f_t \times (bw_{thr} - r_t)$. Subsection 3.3 explains the heuristic and how it calculates $f_t$ to reflect the the sender's impact using the gradient of the available bandwidth ratio.

## 3.3 The Congestion Control Heuristic

The heuristic in our CC algorithm defines a utilization threshold $u_{thr}$ as mentioned above. Figure 2 shows the state machine of the proposed CC heuristic. The connection starts in the "Initial" state where it applies an exponential increase (the CW doubles every RTT). This state also generates historical data for the ML model to estimate the available ratio of the bandwidth. It continues in this state for a predefined time interval (1 second) or until the RTT becomes significantly high (i.e., $RTT > 3 \times RTT_{min}$ ) or packet drop is detected. Then, it will use the collected history and the ML model to estimate $a_t$ at this time. Based on the value of $a_t$, it will enter one of the three states as shown in Figure 2. For each of the three states (Over Utilized, Under Utilized, and In Range), the heuristic utilizes an algorithm to update the sending rate and the CW. It is intuitive to increase the CW in the Under utilized state (UUS) and decrease it in the over utilized state (OUS). However, the question is how much the increase or decrease should be.

To decide how much the rate should increase/decrease, the sender should consider two factors. The first is the current sending rate $r_t$, where the higher the sending rate the lower the increase (and the higher the decrease), and vice versa. The second is how much this rate contributes to the bottleneck load i.e., $m_t = \frac{r_t}{l_t}$, where $l_t$ is the load on the bottleneck link and $m_t$ is the contribution ratio. Again, the higher the flow contribution the lower the increase and the higher the decrease. The following subsections describe how each state changes the sending rate.

*3.3.1 Under Utilized State .* In this state, each sender tries to increase its rate based on the gradient of $a_t$ with respect to its rate changes. Every RTT, the sender calculates the factor $f_t$ at this time then calls Algorithm 1 to calculate the rate and CW, as shown in Algorithm 2.

To calculate $f_t$, the gradient is used as a measure for the impact of this flow on the bottleneck link. The higher this impact, the higher the contribution of this flow to the link utilization, consequently the lower the rate increment. Figure 3 illustrates the meaning of $S_g$ as a measure of impact. On the right side of the graph, $g_t > 0$ means that the sender has increased the rate and the available bandwidth ratio has also increased in the previous RTT. This indicates that either

**Algorithm 1** Update CW

**Input:**($a_t$,$r_t$ , $u_{thr}, \gamma, rtt, f_t$)

**procedure** ▷ (update the sending rate and CW using the increase factor $f_t$)
  **if** $|u_t - u_{thr}| > \gamma$ **then** ▷ as long as the current utilization is far from the target.
    $bw \leftarrow \frac{r_t}{1-a_t} = \frac{r_t}{u_t}$ ▷ Estimate the available BW assuming the worst case.
    $bw_{thr} \leftarrow u_{thr} \times bw$
    $r_{t+1} \leftarrow r_t + f_t \times (bw_{thr} - r_t)$
    $r_{t+1} \leftarrow max(r_{t+1},\ r_{min})$
    $CW \leftarrow r_{t+1}/rtt$
  **end if**
  **return** $CW$ ▷ Return the new sending rate.
**end procedure**

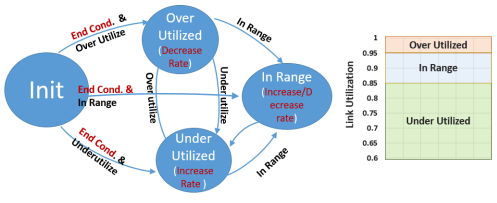



**Figure 2: The state machine and bottleneck utilization.**

the sending rate is very small compared to the other senders, or the other flows on this link have stopped or decreased their rates. In both cases, the sender can safely increase its rate. On the other hand, when $g_t < 0$, this means the $a_t$ changed proportionally to the rate change in the previous interval, which indicates that this flow has a significant impact on the link utilization and the sender should increase the rate inversely with this impact (i.e., if the sender rate is a significant portion of the link load, the rate should be increased very slowly). The factor $f_t$ is calculated as $f_t = f \times (1 - S_g)$ for this situation. In this Algorithm, $f$ is the aggressiveness parameter which is set to 1.25.

**Algorithm 2** Calculate the factor $f_t$ and Update CW

**Input:**($a_t$, $a_{t-1}$,$r_t$, $r_{t-1}$, $S$)

1: **procedure** ▷ (Use the connection statistics ($a_t$, $a_{t-1}$,$r_t$, $r_{t-1}$) and the current state $S$ to calculate $f_t$ and update the CW )
2: $f \leftarrow 1.25$ ▷ Set the aggressiveness parameter $f$
3: $\delta R \leftarrow \frac{r_t - r_{t-1}}{r_t}$
4: $g_a \leftarrow \frac{a_t - a_{t-1}}{\delta R}$ ▷ Calculate the gradient of $a_t$
5: $S_g \leftarrow \frac{1}{1+e^{-g_a}}$ ▷ Calculate the gradient sigmoid $S_g$
6: **if** $S_g \leq 0.5$ **then** ▷ If this flow has impact
7: $\bar{S_g} \leftarrow 2 \times S_g$ ▷ Normalize $S_g$
8: **if** *S is Under Utilize* **then**
9: $f_t \leftarrow f \times \bar{S_g}$ ▷ Update $f_t$
10: **end if**
11: **if** *S is Over Utilize* **then**
12: $f_t \leftarrow f \times (1 - \bar{S_g})$ ▷ Update $f_t$
13: **end if**
14: **end if**
15: **Call Alg. 1 with** $f_t$ ▷ Call Alg. 1 and pass $f_t$
16: **end procedure**

*3.3.2 Over Utilized State .* If the bottleneck utilization $u_t > u_{thr} + \gamma$ then there is a high risk of overwhelming the bottleneck and its queue, which will increase the queuing delay and packet drop. In this state, the sender reduces the congestion window, lowering the sending rate as well. To do that, the sender uses the same Algorithm 2. The only difference between the Under Utilized and the Over Utilized states is in Line 9 and Line 12 in Algorithm 2. In the OUS, the aggressiveness of the decrease should be inversely proportional to $\bar{S_g}$. For instance, if the normalized gradient $\bar{S_g} \rightarrow 1$, this sender has a very low contribution to the bottleneck load and the sender should very slowly decrease its rate; consequently, $f_t$ should be close to zero which will be multiplied by a negative value ($bw_{thr} - r_t$) in Algorithm 2.

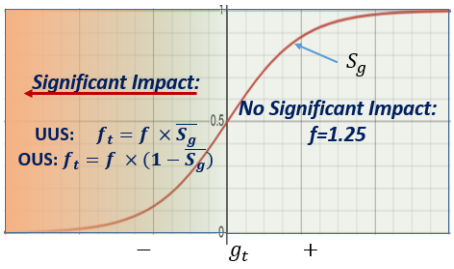



**Figure 3: The gradient as a measure of impact.**

*3.3.3 In Range State .* In this state, each sender tries to increase its rate as a kind of probing for available bandwidth. If all senders increase their rates by the same amount, the sender with the highest rate will dominate the bandwidth. To achieve a fair bandwidth share among different senders, each sender increases its rate while considering two factors. The first is the current sending rate $r_t$, where the increase should be inversely proportional to its current rate. The second factor is the impact of the current rate on the link utilization, similar to the UUS and OUS.

The first factor is considered by calculating $S_r$ which is the sigmoid of the current rate divided by , the rate impact parameter, and normalizing it (since $S_r \in [0.5, 1]$) as in Equation 1. The rate is then increased by a factor of $1 - S_r$ as in Equation 2. This way, the senders with high rates will increase their rates very slowly while senders with the slow rates can quickly increase their rates.

$$S_r = 2 \times \left(\frac{1}{1 + e^{-\frac{r_t}{\beta}}} - 0.5\right) \quad (1)$$

$$r_{t+1} = r_t + (1 - S_r) \quad (2)$$

where $\beta = 1$ is the rate impact parameter. This parameter can also be used to control how aggressive the sending rate changes. Increasing $\beta$ can lead to a faster convergence but with that comes a risk of rate and delay fluctuation.

To consider the sender's impact, which can be measured by the gradient, the sender applies the UUS procedure. Doing this allows the algorithm to address very extreme cases such as low link capacities, in which even small rates may have a significant impact.

## 3.4 The Prediction Model

Estimating the ratio of the bottleneck link utilization is an important component of the proposed MLACC. In this section, we introduce our ML prediction model that takes in historical network statistics and outputs the available bandwidth ratio in the network.

*3.4.1 The Available Bandwidth vs. Its Ratio.* Theoretically, it may be easier for the CC algorithm to adjust its sending rates with an available bandwidth prediction rather than the bandwidth ratio. However, the prediction of the absolute available bandwidth is dependent on the total link capacity and the current load(s) on the link. This makes training a model extremely challenging since it would require more features than those we can obtain from connection statistics. Moreover, the prediction accuracy will be significantly low due to the weak relationship between the connection statistics and the actual value of the total link capacity. On the other hand, the ratio of the available bandwidth to the total link capacity can be directly translated to the link utilization, which is highly correlated to the end-to-end latency (or queuing delay) and packet drop rates, making the prediction of available bandwidth ratio easier and more accurate.

*3.4.2 Data Collection.* Acquiring a ground truth bandwidth ratio is among the most critical and difficult factors in producing an accurate bandwidth ratio prediction. It is extremely difficult to measure the available bandwidth within real networks along with end point statistics. To gather accurate data for training the ML model, NS3

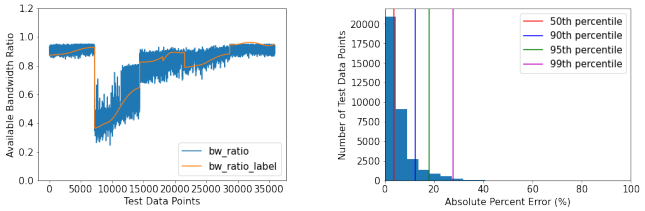
[9] is used. Employing a simulator gives the flexibility of creating different network environments and conditions whilst collecting statistics from both network devices (bottleneck link utilization) and end nodes (input features). The set up for data collection is as follows.

On the NS3 simulation platform, we created a network topology shown in Figure 5. The capacity of the bottleneck link and its queue size are changed to model different real-world network environments with queuing delay and packet loss. Using a simulated environment also greatly reduces the data generation time compared to an emulation approach, which gives us the opportunity to produce data with link capacities ranging from 5Mbps to 50Mbps. Over this topology, a client is configured to send background traffic to the server with normally distributed packet size and random exponentially distributed packet inter-arrival time to mimic background traffic in a real network. The added randomness in the packets help prepare the model for uncertainty in a real network. Simultaneously another client sends probing data to measure the connection parameters by recording packet sending/receiving times and sequence numbers to calculate the average and standard deviation of latency, its first and second order derivatives, and packet loss. These parameters are the input features used in the bandwidth ratio estimation. The features are smoothed using sliding window to reduce the effects of random sudden fluctuations in the individual measurements. The ground truth bandwidth ratio is calculated by dividing the available bandwidth by the link capacity. Both the input features and the ground truth are then time-matched and sorted as time-series data in preparation to be processed by the ML model.

*3.4.3 Model Architecture.* The architecture is a LSTM model composed of multiple layers of LSTM units each containing a memory cell and three gates: input gate, forget gate, and output gate. The memory cell, or cell state, carries information through the model which can be added or removed from the cell by the gates. These gates control what the model memorizes, and this is done with the help of sigmoid activation functions [7].

The input to our LSTM model is packet-level historical network statistics with a resolution of 0.1 seconds. We use the $n$ most recent time-steps as our input window $[t-n, t]$, and the next $t$-th time-steps as our output prediction window where $t = \{n, n+1, n+2, ...\}$. The historical data is represented by $X = [x_{t-n}, ..., x_{t-1}]$ such that $x \in \mathbb{R}^{m \times n}$ where $m$ is the number of features. Since the LSTM functions by remembering patterns of the input data, a historical length of 10 time-steps is set to give enough past information to the LSTM while still maintaining its adaptability to fluctuations in the network. The architecture of our model includes one input layer, three hidden layers with 10 LSTM units each, and one output layer. The internal parameters of the model used to map the input features to the output prediction are updated during the training stage and a grid search is conducted to select the best model hyperparameters.

*3.4.4 Model Prediction Results.* Once the simulated data has been sorted into a time matched data format for each link capacity, each individual set of data is then divided using an 80/10/10 split into three groups: train, validation, and test. The divide is done randomly so that the model is not always training and tuning its parameters on data of the same pattern and overfitting. The data sets are then normalized before inputting to the model.




**(a) Predicted vs. Actual Values** **(b) Prediction Error Distribution**

**Figure 4: The prediction results.**

After training and parameter tuning using validation data, the performance of the model is evaluated on unseen test data. The main metric used to quantify the test prediction results is the mean absolute percentage error (MAPE).

Our model was able to achieve an overall 4.15% MAPE and as can be seen in Figure 4a, it is able to differentiate the bandwidth ratios across 5 different link capacities. Aside from the mean error value, the distribution of the absolute percent error was also taken into consideration (Figure 4b). By calculating the error percentiles, it was found that almost 65% of our predictions lie within the 5% MAPE range, and only less than 4% are above a 20% error rate.

## 4 RESULTS AND EVALUATION

To evaluate the proposed CC framework, we implemented it in NS3 simulator [9]. We use the topology in Figure 5 to compare the performance of the proposed CC technique to that of TCP-BBR with different clients sending data to the server. We elected to compare with BBR because it is implemented in several operating systems and it is on par with the SOTA techniques [19]. The access links (between R1 and the clients and between R2 and the server) are configured with $1Gbps$ speed and $5ms$ latency. The latency of the bottleneck link is set to $10ms$ , while its capacity $C$ takes different values [5Mbps, 10Mbps, 15Mbps] to study different scenarios. In all of the experiments performed, $u_{thr} = 0.95$ and $\gamma = 0.05$.

We started the evaluation using a simple scenario where one client sends a stream of data to the server for 10 seconds. The average total throughput and the average end-to-end (E2E) latency over the $10\ sec$ are calculated and presented in Table 1 which shows the significant reduction in the E2E latency. It also illustrates that the throughput of MLACC is reduced by a small ratio. However, this reduction becomes insignificant as the capacity increases. The throughput of the proposed technique can be improved by increasing the link utilization threshold $u_{thr}$, however, this will increase the risk of packet drop and increase the latency.

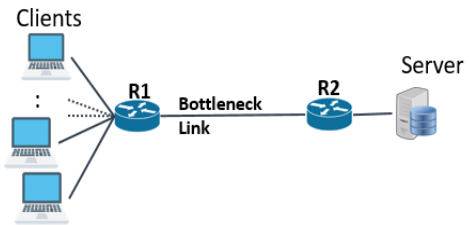
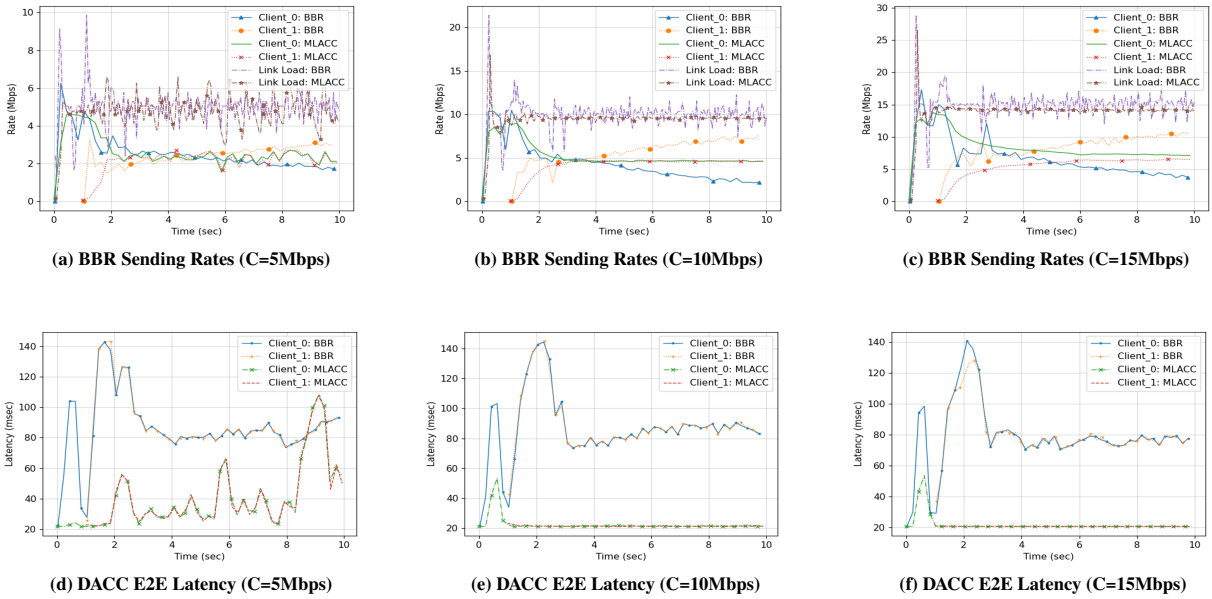



**Figure 5: Evaluation Topology**

, ,




(a) BBR Sending Rates (C=5Mbps)

(b) BBR Sending Rates (C=10Mbps)

(c) BBR Sending Rates (C=15Mbps)

(d) DACC E2E Latency (C=5Mbps)

(e) DACC E2E Latency (C=10Mbps)

(f) DACC E2E Latency (C=15Mbps)

**Figure 6: BBR vs MLACC**

To study how different connections impact the bottleneck, we ran an experiment in which two clients send data to the server. The first client starts at time zero and the other starts after 1 second. Figure 6 compares the sending rates and the E2E latency for three scenarios of the bottleneck capacity. It shows that the E2E latency is reduced by a range between 50% to 75% compared to BBR, as shown in Figure 6d, 6e, 6f. This also means that the packet drop is reduced because of the shorter queuing delay. Figure 6 also demonstrates the fair sharing of the link capacity in the case of MLACC, as this is an important design objective. These benefits are achieved by sacrificing a small portion of link utilization less than 3.5% in the worst case. Although there are times, especially in lower bandwidth scenarios, where the latency of MLACC becomes higher than BBR (Figure 6d), further investigation demonstrated that the reason is the prediction error, which is inevitable with ML approaches. Even with this prediction error, MLACC continues to outperform the BBR algorithm. The effect of the prediction error also becomes negligible in cases of higher network capacities.

Figure 6c shows an important shortcoming in MLACC where its convergence becoming slow as link bandwidths increase. Although BBR has a slight advantage in convergence time, a fair share can not be guaranteed between different BBR connections while MLACC can achieve this goal. The proposed model uses two factors to control the convergence, namely the aggressiveness factor $f = 1.25$ and the rate impact factor $\beta = 1$. The results are based on these values. We can increase these factors to achieve faster convergence, but this may increase the sending rate fluctuation especially in cases of low bandwidth which is a well known trade-off between convergence and stability.

**Table 1: BBR vs. MLACC for one client connection**

| | 5Mbps | | 10Mbps | | 15Mbps | |
|---|---|---|---|---|---|---|
| | **Av. Rate** | **Av. Lat.** | **Av. Rate** | **Av. Lat.** | **Av. Rate** | **Av. Lat.** |
| **BBR** | 4.82 | 30.34 | 9.04 | 44.94 | 13.60 | 36.07 |
| **MLACC** | 4.67 | 22.51 | 9.14 | 21.49 | 13.65 | 21.82 |
| **Saving %** | -3.19 | 25.80 | -1.12 | 52.17 | -0.31 | 39.50 |

## 5 CONCLUSION

In this paper, we present a new congestion control algorithm, MLACC, that combines heuristics with a ML model and aims to minimize the queuing delay and packet loss rates at the bottleneck link while maintaining fairness across different connections that are sharing the bottleneck. The proposed solution utilizes a ML model to estimate the link utilization. The heuristic uses it to adjust the sending rate. The proposed CC algorithm achieves high fairness as well as low and stable end-to-end latency and drop rates by sacrificing less than 3.5% of the throughput. This paper presents a first step toward efficient and fair CC algorithm that can satisfy the requirements of the emerging inter-cloud parallel applications. However, there are several extensions can be conducted based on this step. An important extension is to mathematically optimize the parameters used by the heuristic, the aggressiveness factor $f$ and the rate impact factor $\beta$, or utilize another machine learning technique to optimize them such as DRL. We also intend to compare MLACC to alternative techniques that employ a similar approach and perform extensive analysis for a wider range of network settings. We also plan to study the performance of inter-cloud parallel computation applications under the proposed congestion control technique.

# REFERENCES


[1] Stephen Bensley, Dave Thaler, Praveen Balasubramanian, Lars Eggert, and Glenn Judd. 2017. *Data center TCP (DCTCP): TCP congestion control for data centers*. Technical Report.

[2] Abdelhak Bentaleb, Mehmet N. Akcay, May Lim, Ali C. Begen, and Roger Zimmermann. 2022. BoB: Bandwidth Prediction for Real-Time Communications Using Heuristic and Reinforcement Learning. *IEEE Transactions on Multimedia* (2022), 1–16. https://doi.org/10.1109/TMM.2022.3216456

[3] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *SIGCOMM Comput. Commun. Rev.* 24, 4 (oct 1994), 24–35.

[4] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. 2019. When to Use and When Not to Use BBR: An Empirical Analysis and Evaluation Study. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 130–136.

[5] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control: Measuring Bottleneck Bandwidth and Round-Trip Propagation Time. *Queue* 14, 5 (oct 2016), 20–53.

[6] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (jan 2008), 107–113.

[7] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to Forget: Continual Prediction with LSTM. *Neural Computation* 12, 10 (2000), 2451–2471.

[8] Oussama Habachi, Hsien-Po Shiang, Mihaela van der Schaar, and Yezekael Hayel. 2013. Online Learning Based Congestion Control for Adaptive Multimedia Transmission. *IEEE Transactions on Signal Processing* 61, 6 (2013), 1460–1469. https://doi.org/10.1109/TSP.2012.2237171

[9] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. 2008. Network simulations with the ns-3 simulator. *SIGCOMM demonstration* 14, 14 (2008), 527.

[10] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. 1–10. https://doi.org/10.1109/ICNP.2017.8117540

[11] V. Jacobson. 1988. Congestion Avoidance and Control. *SIGCOMM Comput. Commun. Rev.* 18, 4 (aug 1988), 314–329.

[12] Vivek Jain, Viyom Mittal, and Mohit P. Tahiliani. 2018. Design and Implementation of TCP BBR in Ns-3. In *Proceedings of the 2018 Workshop on Ns-3* (Surathkal, India) *(WNS3 '18)*. Association for Computing Machinery, New York, NY, USA, 16–22.

[13] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 3050–3059.

[14] Huiling Jiang, Qing Li, Yong Jiang, GengBiao Shen, Richard Sinnott, Chen Tian, and Mingwei Xu. 2021. When machine learning meets congestion control: A survey and comparison. *Computer Networks* 192 (2021), 108033. https://doi.org/10.1016/j.comnet.2021.108033

[15] Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. 2018. *CUBIC for fast long-distance networks*. Technical Report.

[16] Yeong-Jun Song, Geon-Hwan Kim, Imtiaz Mahmud, Won-Kyeong Seo, and You-Ze Cho. 2021. Understanding of bbrv2: Evaluation and comparison with bbrv1 congestion control algorithm. *IEEE Access* 9 (2021), 37131–37145.

[17] Nikko Ström. [n.d.]. Scalable Distributed DNN Training Using Commodity GPU Cloud Computing.

[18] Lizhuang Tan, Wei Su, Wei Zhang, Jianhui Lv, Zhenyi Zhang, Jingying Miao, Xiaoxi Liu, and Na Li. 2021. In-band network telemetry: A survey. *Computer Networks* 186 (2021), 107763.

[19] Wenting Wei, Huaxi Gu, and Baochun Li. 2021. Congestion Control: A Renaissance with Machine Learning. *IEEE Network* 35, 4 (2021), 262–269. https://doi.org/10.1109/MNET.011.2000603

[20] Keith Winstein, Anirudh Sivaraman, Hari Balakrishnan, et al. 2013. Stochastic forecasts achieve high throughput and low delay over cellular networks.. In *NSDI*, Vol. 1. 2–3.

[21] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, San Jose, CA, 15–28.

[22] Han Zhang, Xingang Shi, Xia Yin, Fengyuan Ren, and Zhiliang Wang. 2015. More load, more differentiation—A design principle for deadline-aware congestion control. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 127–135.

[23] Hao Zhang, Haiting Zhu, Yu Xia, Lu Zhang, Yuan Zhang, and Yingying Deng. 2019. Performance Analysis of BBR Congestion Control Protocol Based on NS3. In *2019 Seventh International Conference on Advanced Cloud and Big Data (CBD)*. 363–368. https://doi.org/10.1109/CBD.2019.00071

[24] Lei Zhang, Kewei Zhu, Junchen Pan, Hang Shi, Yong Jiang, and Yong Cui. 2020. Reinforcement Learning Based Congestion Control in a Real Environment. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 1–9. https://doi.org/10.1109/ICCCN49398.2020.9209750