BBR-FIT: An Intelligent BBR based on the Reinforcement Learning to Boost the Network Efficiency over Time-Varying Networks

1st YiXie
Ningbo University
Ningbo, China
2011082082@nbu.edu.cn

2nd Xianliang Jiang
Ningbo University
Ningbo, China
jiangxianliang@nbu.edu.cn

3rd Guang Jin
Ningbo University
Ningbo, China
jinguang@nbu.edu.cn

4rd Haiming Chen
Ningbo University
Ningbo, China
chenhaiming@nbu.edu.cn

***Abstract*—Recently, the emerging high-speed and low-latency communication techniques (e.g., 5G and WiFi6) reactivate the popularity of some Internet applications. Most of them simultaneously need the support of high-throughput and low-latency transmission provided by underlying networks. As the core component to ensure reliable data delivery, TCP congestion control algorithms have attracted the attention of Linux kernel developers over the past two decades. BBR is a representative congestion control algorithm developed by Google in 2016. Compared with traditional algorithms, BBR probes the maximum available bottleneck bandwidth and minimum Round-Trip Time. This mechanism makes BBR perform better than previous schemes over high bandwidth-delay and lossy networks. However, according to the extensive measurement of BBR over China Mobile 5G networks, we found that the performance of BBR is not good when the network state changes frequently. It is mainly caused by the fixed gain parameter. To solve this issue, we adopt the wisdom of reinforcement learning techniques to fine-tune the gain parameter and provide an intelligent BBR (named BBR-FIT). Extensive experiments with the mahimahi tool and real testbed indicate that, compared with BBR, BBR-FIT can promote the link utilization (2×) without inducing an increase in packet delay.**

***Index Terms*—BBR, reinforcement learning techniques, fine-tune the gain parameter, dynamic networks**

I. Introduction

Real-time and interactive multimedia such as live-streaming and virtual/augmented reality have been gaining popularity all over the world. With enhanced mobile broadband (eMBB) as the predominant scenario for 5G networks, the demand for real-time streaming applications is expected to be met [1]. On the one hand, the new wave of video quality technology innovations causes real-time streaming applications, such as 4K video streaming, to consume more network bandwidth [2]. On the other hand, some interactive applications, like video conferencing, require transmission latency close to real-time.

Manuscript received xx xx, 2022. This work was supported in part by the National Natural Science Foundation of China (61601252), the Natural Science Foundation of Zhejiang Province (LY20F020008, LY21F020006), the Ningbo Natural Science Foundation (202003N4085), the Ningbo Public Welfare Project (202002n3109), the Special Research Funding from the Marine Biotechnology and Marine Engineering Discipline Group in Ningbo University (No. 422004582), the Ningbo Key Science and Technology Plan (2025) Project (2019B10125, 2019B10028, 20201ZDYF020077).

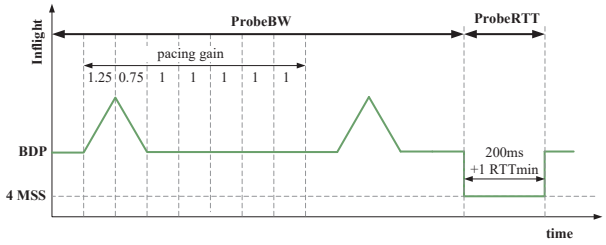


Fig. 1. The essential working model of BBR.

As the fundamental component for ensuring reliable data delivery, TCP congestion control algorithms (CCAs) have attracted the interest of Linux kernel developers over the past two decades. The primary objective of traditional CCAs is to enhance the performance in specific networks [3] [4] [5]. For instance, CUBIC [6] was designed for wired networks and is typically inapplicable in cellular networks. Each year, a new wave of technological innovations would inspire a new wave of CCA design. At least 15 CCAs have been implemented in the Linux kernel thus far. Therein, BBR [7] is an example of a CCA developed by Google in 2016. BBR probes the maximum available bottleneck bandwidth and minimum Round-Trip Time (RTT). Fig. 1 depicts the essential working model of BBR. Compared with loss-based or delay-based CCAs, BBR claims to be a rate-based CCA. Specifically, BBR initiates the CCA mechanism at the optimal operating point by periodically measuring the minimum RTT and maximum bandwidth. The product of these two parameters is known as the bandwidth product (BDP).

5G networks introduce the small-cell concept, making it impossible for 5G base stations to cover vast regions. Therefore, the 5G capacity may be highly jittered during the fast movement [8]. The increment in traffic further exacerbates the capacity jitter in 5G networks. As the default CCA for Google services, BBR commands 11% of US Internet traffic [9]. However, simulations in Section II demonstrate that BBR has a severe performance degradation in time-varying networks. We explore the reasons behind this problem in the next

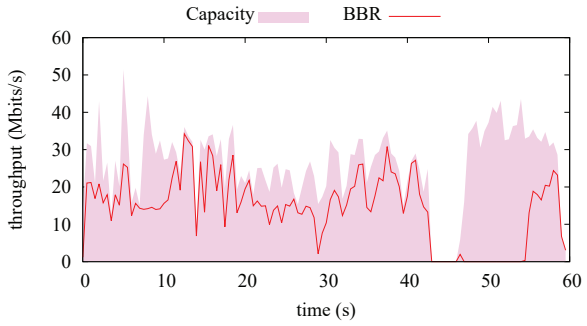


Fig. 2. Throughput of BBR over time in 5G networks.

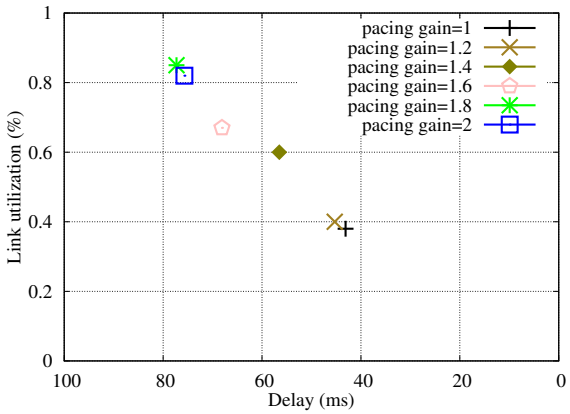


Fig. 3. Throughput and delay for different pacing gain in steady phase.

section. Briefly, BBR does not detect the change in network capacity in time. Our objective is to enhance the efficiency of BBR in time-varying networks, such as China Mobile's 5G networks. To solve these issues, we present BBR-FIT in this paper. Unlike the original BBR, BBR-FIT is an intelligent BBR based on reinforcement learning (RL) to boost network efficiency over time-varying networks. After training, BBR-FIT can automatically adjust the pacing rate. The summary of our contributions is as follows:

- We discover the performance degradation of BBR in time-varying networks.
- We propose a novel CCA, BBR-FIT, that adjusts the pacing gain in the steady phase by employing RL techniques.
- We implement BBR-FIT based on the Mahimahi simulator, deploy it with the Linux Kernel 4.13.10, and compare it to other CCAs. The results demonstrate that BBR-FIT outperforms other CCAs regarding throughput-latency tradeoff in various network scenarios.

II. Motivation

By introducing millimeter-wave (mmWave) communications, 5G networks offer the potential multi-gigabit-per-second rate for real-time applications. However, mmWave capacity can be highly dynamic (Fig. 2). [11] demonstrates that manually tuned TCP flow performs poorly predicting bandwidth. Today's networks are in a highly dynamic and complex environment. The decision-making process of manually tuned TCP CCAs is based on the pre-defined rules designed by human understanding of the network. Since human knowledge does not always accurately characterize the network features, manual pre-defined rules can only help manually tuned CCAs achieve sub-optimal performance. Moreover, manually tuned CCAs assume no prior information about networks and fail to learn from historical knowledge. Despite many years of improvements, manually tuned CCAs still suffers from an unsatisfactory performance [12], [13]. With the emergence of several dedicated libraries, such as TensorFlow, Caffe, and PyTorch, building an accurate model for RL has been dramatically simplified. RL techniques would help manually tuned CCAs learn network information from experience.

To evaluate the performance of BBR in time-varying networks, We employ the topology and 5G trace in Section IV. As shown in Fig. 2, the purple background part shows the link capacity over time, while the red line is the BBR throughput over time. BBR underutilizes the bottleneck bandwidth in 5G networks. BRB probes the bandwidth in the first two RTTs of the ProbeBW phase and keeps the pacing gain constant in the last six RTTs (so-called steady phase). [14] points out that this model is the main reason for the throughput degradation of BBR in time-varying networks.

We consider the worst scenario for this model, where the bandwidth rises at the start of the steady phase. To simulate such a scenario, We employ Mahimahi to simulate a network with a bandwidth of 100M, a latency of 40ms and a buffer size of 1BDP. A server and a client are connected through a switch. Meanwhile, once we monitor the BBR to enter the steady phase, we use the TC command in a switch to increase the bandwidth and recover it after the steady phase end. Finally, we investigate the impact of pacing gain on BBR performance. Fig. 3 shows the link utilization and delay for different pacing gain. A larger pacing gain is beneficial for sensing bandwidth changes in time while obtaining a large delay. In summary, even in the steady phase, BBR should increase its pacing gain when the bandwidth increases and decrease its pacing gain when the delay increases.

III. BBR-FIT's Design

This section introduces combining RL techniques and the BBR congestion control strategy. The proposed BBR-FIT can automatically decide when and at what pacing rate to probe bandwidth by interacting with the environment. Fig. 4 demonstrates the main framework of BBR-FIT. BBR-FIT consists of three parts: 1-Monitoring block, 2-Underlying BBR logic, and 3-RL agent. Specifically, the Monitoring block collects network state information during a steady phase and collates it into status and rewards to the RL agent. According to these information and policy networks, the RL agent selects a pacing gain for Underlying BBR logic. For the RL agent, we select PPO to citeref:ppo, a classical RL

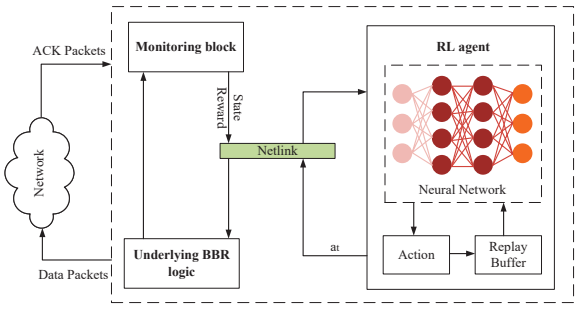


Fig. 4. The main framwork of BBR-FIT.

algorithm, as the decision-maker for congestion control. At the same time, the underlying BBR logic module informs the Monitoring block to collect network state information. During this cycle, the RL agent constantly interacts with the network environment. Therein, Underlying BBR logic inherits the congestion control mechanism of BBR, we no longer go into detail.

A. Monitoring block

The Monitoring block works as an intermediary for BBR-FIT. The RL agent is present at the application layer, while the Underlying BBR belongs to the kernel. Therefore, the RL agent and the Underlying BBR module cannot interact directly. The Monitoring block address this problem by employing Netlink[1], a Linux kernel interface for inter-process communication between the kernel and user-space processes. Through Netlink, the Monitoring block can extract network state information from the Underlying BBR logic block and update the input representing the environment for the RL agent. In detail, after receiving the signal from the underlying BBR logic module, the monitoring block starts monitoring the incoming ACK packets to collect the required packets' statistics. The statistics we considered are shown in Table I.

TABLE I
STATISTICS GENERATED BY MONITORING MODULE

Variables	Comments
s_r	The ratio of sending rate to receiving rate
l_r	The average loss rate of packets
l	The average latency of packets
l_m	The minimum value of packet delay during probe cycle
t	The average throughput
d	the time of a probe cycle
l_i	the derivative of latency with respect to time

B. RL agent

The core of BBR-FIT is the RL agent. With the historical environmental statistics generated by the monitoring module in the Replay Buffer, the RL agent selects the appropriate pacing gain for the Underlying BBR logic module to maximize its reward. Like other typical RL problems, the RL agent for BBR-FIT consists of State, Action, and Reward.

1) Action: In BBR-FIT, the RL agent takes action to specify how the Underlying BBR logic module should adjust its pacing gain in response to the observed network state. Like the original BBR, BBR-FIT aims to work at the optimal Kleinrock's point, to achieve high throughput and low latency. Inspired by [16], We express the action as changes to the current rate and map the RL agent's action based on Equation (1) where a is 0.025 for dampening oscillations.

$$p_t = \begin{cases} p_{t-1} * (1 + \alpha \times a_t) & a_t \geq 0 \\ p_{t-1} / (1 - \alpha \times a_t) & a_t < 0 \end{cases} \quad (1)$$

2) State: After the RL agent selects the action for the Underlying BBR logic module, the Monitoring module observes the results generated by this action. It calculates them as a state vector v_t. A reasonable state space can help the RL agent efficiently achieve its goals. Network environments can be complicated, with dramatic bottleneck bandwidth and latency changes. Only with enough information can the RL agent accurately capture the performance of its actions. Many state variables can characterize the network environment. For instance, the mean interval between packets sent means the interval between ACKs, the number of unacknowledged packets, etc. Too many state variables would cause an increase in computation cost and significant delay convergence. Therefore, Appropriate state variables can help BBR-FIT capture the performance of actions it takes and ensure a manageable learning process. Based on the above discussion, We choose a state space that includes the following parameters:

s: Ratio of sending rate to receiving rate. we consider s is essential to determine whether the network is volatile.

l_i: Derivative of latency with respect to time. l_i can reflect the fluctuation of the latency.

l_r: Ratio of current latency to minimum average latency. It directly reflects the degree of network congestion.

The RL agent's selection of the next action was learned from historical knowledge. We store a fixed-length history of the above statistical vectors in the Replay Buffer. A bounded-length and appropriate history enable our agent to make responses that match the trends in network conditions. We consider the history length used in Section III-C2 and choose

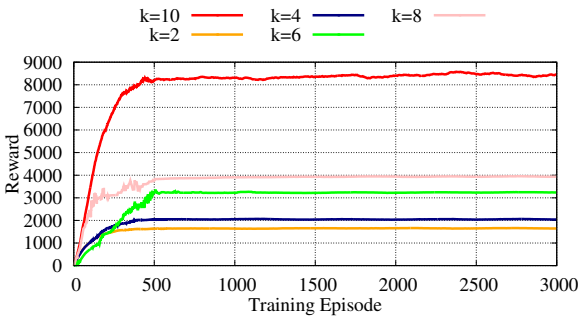


Fig. 5. Training reward for RL agent.

[1]http://netfilter.org/projects/libmnl

our history length set to 10. the state at time t, s_t, is defined in Equation (2). d is the delay between selecting a pacing rate and collecting the results.

$$s_t = (v_{t-(10+d),\ldots,v_{t-d}}), \tag{2}$$

3) reward: The reward function specifies the goal of BBR-FIT. Like other RL CCA, BBR-FIT consider throughput (t), latency (l) and loss rate (l_r) as metrics. [21] pointed out that there is a significant degradation in the performance of RLCC when only one of the metrics is present in the reward function. Moreover, when all three metrics in the reward function are considered simultaneously, the protocol has the desired performance. Therefore, we design the following utility function:

$$reward = \alpha \times t - \beta \times l - \gamma \times l_r. \tag{3}$$

Therein, α, β, and γ represent the impact factor assigned to the above metrics. Taking different impact factor combinations can achieve different application performance requirements. A larger value of α meets the application's need for high throughput. While a larger value of β meets the application's need for low latency, a larger value of γ ensures a low retransmission rate of the video transmission. To make the BBR-FIT more versatile and accommodate various network conditions, we use [21]'s best points in minimum delay and maximum throughput, i.e.,α=0.2,β=0.6 and γ=0.2.

C. Learning algorithm

We employ Mahimahi to simulate a time-varying network environment to train our RL agent. The training algorithm is devoted to maximizing the long-term reward for the RL agent. In BBR-FIT, the training algorithm is PPO, which can be invoked with the stable-baselines python package.

1) BBR-FIT's Learning Algorithm: The goal of the RL agent is to select optimal action for a specific state to maximize learning reward behavior. During the training process of BBR-FIT, the neural network produces the policy for p_t adjustment to map the current network state to the p_T to maximize the tradeoff between throughput and latency. During time interval t, depending on the observed state s_t and its policy π_θ, the agent selects the action to obtain the reward r_t. The eventual aim is to find the optimal policy π, with parameters θ, to maximize the total reward $J_{PPO}(\theta) = E[min(r(\theta)\hat{A}_t, clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$.

The agent algorithm of BBR-FIT is not only based on the actor-critic framework but also belongs to the Policy Gradient algorithm. It contains Critic (action-value function) and Actor (new and old policy) parameterized by deep neural networks. Actor uses new policy to interact with the network environment and records a sample $\tau_i = \{s_t, a_t, r_t\}$ in Replay buffer at each timestep t. After T timesteps, a set of trajectories $D_k = \{\tau_i\}$ are collected. After that, Actor copies the new policy's parameters into the old one. The new policy and Critic are learned using the data in the replay buffer. The new policy update parameters are as follows:

Algorithm 1 BBR-FIT's Learning Algorithm

Input: Initialize policy π with parameters θ_0, initialize value function parameters ϕ_0;
for $k \in \{1, 2, \ldots\}$ **do**
  Run policy $\pi_($ $\theta_k)$ for T timesteps:
    Select a_t from $\pi_($ $\theta_k)$
    Execute p_t to the steady phase according to Equ (1)
    Observe the new states s_{t+1} and the reward r_t;
    Store sample $\tau_i = \{s_t, a_t, r_t\}$ into replay buffer
  Collect set of trajectories $D_k = \{\tau_i\}$
  Compute advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_t - V_\phi(s_t)$;
  $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$
  $\pi_{old} \leftarrow \pi_\theta$
  Update the policy by maximizing the objective function $J_{PPO}(\theta) = E[min(r(\theta)\hat{A}_t, clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$
  Update policy parameters θ_{k+1} with Equation 4
  Update value function parameters ϕ with Equation 6
end for

$$\theta_{k+1} = \underset{\theta}{argmax} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} min(r(\theta) A^{\pi_{\theta_k}}, g(\in, A^{\pi_{\theta_k}})). \tag{4}$$

$$\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_t - V_\phi(s_t). \tag{5}$$

Where $A^{\pi_{\theta_k}}$ is updated by critic according Equation (4). Equation (6) denotes the process of Critic to update rules.

On-policy utilizes only the best short-term options and does not maximize long-term returns. BBR-FIT solves this issue by training the new policy with the old policy. As made in the PPO, BBR-FIT imposes the constraint by forcing $r(\theta)$ to stay within a small interval around 1, precisely $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter.

$$\phi_{k+1} = \underset{\phi}{argmin} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2. \tag{6}$$

2) training: We stitch the traces in [22] into one trace and simulate the time-varying network environment with the Mahimahi simulator. BBR-FIT's agent runs on a PC (CPU: Intel i7-9700 3.00GHz×8; Memory: 16GB; OS: 64-bit Ubuntu 16.04 with Linux kernel 4.13.10). We start training round by round—a history length k impact the performance of the RL agent. We train models with a history length varying from 2 to 10. Fig. 5 shows the reward of these models. The model obtains the greatest reward and convergence rate for k = 10.

IV. Evaluation

A. Experiment setup

We implement the proposed BBR-FIT in Linux 4.13.10. We emphasize that BBR-FIT can automatically learn the correct

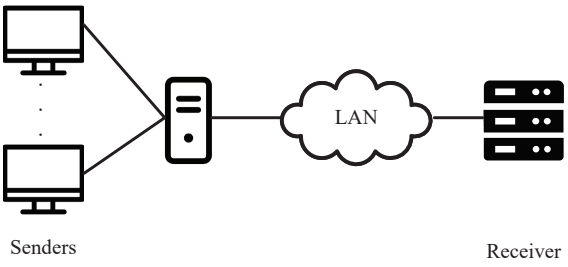


Fig. 6. The topology used for evaluation.

$pacing\_rate$ changing strategy for high throughput and low latency in time-varying networks.

1) Topology: The network topology used for evaluation is shown in Fig. 6, consisting of Senders, trace-driven simulator (Mahimahi), and Receiver. The Mahimahi employ traces to reproduce network conditions. We run "iperf -Z CCA" for 100s on one of the Senders and analyze the Mahimahi logs to obtain the performance of CCA. For these evaluations, unless mentioned, the network parameter settings follow Table II.

2) Traces: Here, we employ the traffic generation tool Saturator[2] to collect real-world traces. For covering a wide range of network scenarios, time-varying networks and stable networks are considered the two main scenarios. For time-varying networks, we collect traces while riding the subway in the center of Ningbo city. We generate four traces for these evaluations: 1. 4G trace 2. 5G trace 3. a stable network with 24Mbps throughput. 4. wifi trace.

TABLE II
THE SIMULATION SETTINGS

Parameters	Value
$SimilationTime$	100s
$Bandwidth$	100Mbps
RTT	40ms
$PacketSize$	1500Bytes
$ACKSize$	40Bytes
$PacketLossRate$	0%
$BufferSize$	1BDP

3) Baseline algorithms: In this section, we compare proposed BBR-FIT with various CCAs: CUBIC [6], PCC [16], and variants of BBR covering different solutions. Among them, the performance of CUBIC and PCC has been proven through extensive literature. While the variants of BBR, including BBR v2 [17] (Solve the problem of BBR generating a lot of packet loss, deployed on Linux 5.4.6), Stateful-BBR [18](Addressing BBT throughput degradation in wireless networks, deployed on Linux 5.4.6), Tsunami[3](Improving the performance of BBR in time-varying networks, deployed on Linux 4.13.10), TCP D* [19](BBR using cwnd instead of pacing rate for congestion control, deployed on Linux 5.4.6), BBRPlus[4] (Improving the performance of BBR in time-varying networks, deployed on Linux 4.19) and BBR-ACD [20] (enhancing the internal-fairness of BBR, deployed on Linux 4.13.10). In these evaluations, we use two main performance metrics: average throughput and average latency.

[2]https://github.com/keithw/multisend/blob/master/sender/saturatr.cc
[3]https://github.com/dlxg/Linux-NetSpeed

B. Performance analysis

Fig. 7 shows the experimental results using different traces in the Mahimahi simulator. There are six figures, each with the horizontal axis indicating the latency and the vertical axis indicating the average link utilization for each CCA. As the CCA approaches the upward and rightward regions of the graph, it indicates that the CCA can achieve higher latency and throughput.

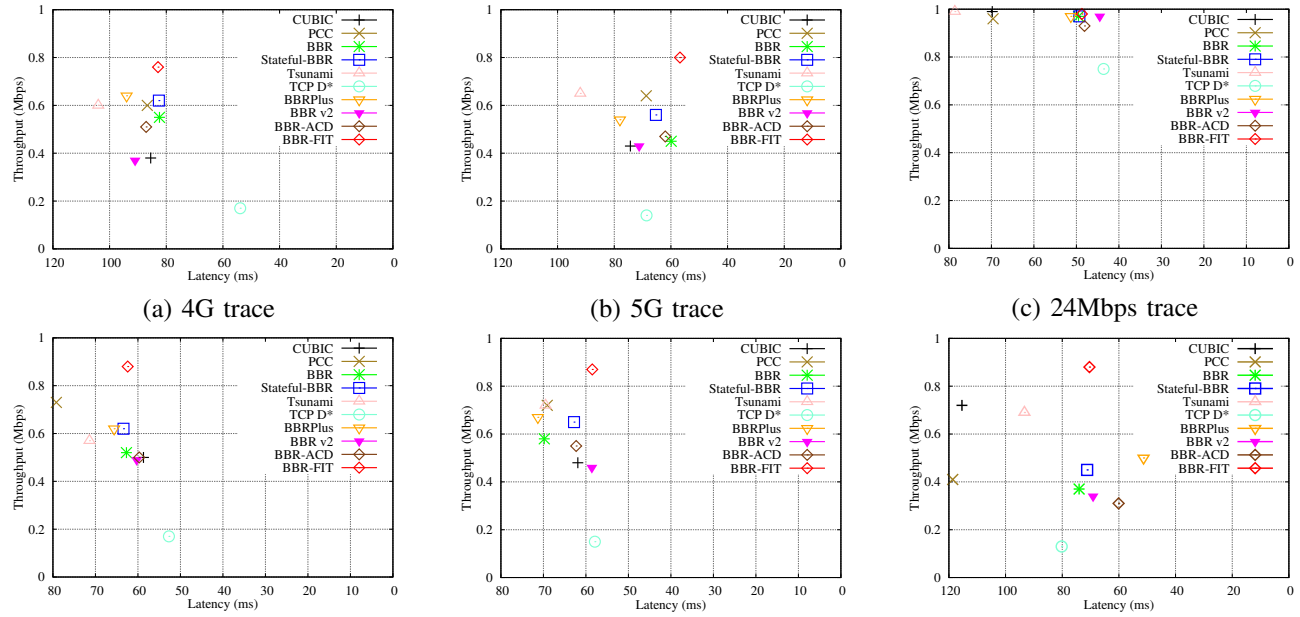
Table III shows the averaged results of the various CCAs through all traces. BBR-FIT achieves the highest tradeoff between latency and throughput among all CCAs. Highlights include: BBR-FIT has 1.5×, 1.3×, and 1.5× higher average link utilization compared to CUBIC, BBR, and BBR v2, respectively. Moreover, compared to TCP D*, which achieves the lowest latency, BBR-FIT increases the average link utilization by about 4× while it only multiplies the latency by 0.28×.

TABLE III
STATISTICS GENERATED BY MONITORING MODULE

Algorithms	Averaged Link Utilization	Averaged Latency
$CUBIC$	0.556	70.04ms
PCC	0.73	74.66ms
BBR	0.614	64.86ms
$Stateful-BBR$	0.684	64.68ms
$Tsunami$	0.706	83.1ms
$TCPD*$	0.276	**55.34ms**
$BBRPlus$	0.688	72.04ms
$BBRv2$	0.544	67.12ms
$BBR-ACD$	0.592	63.84ms
$BBR-FIT$	**0.858**	61.86ms

BBR-FIT improves throughput significantly over others due to its unique adaptive pacing rate mechanism. In this sense, BBRPlus and Tsunami are similar to BBR-FIT because they reduce the length of the probe cycle. However, they are worse than BBR-FIT in latency. The main reason is that BBRPlus randomly reduces the probe phase length without the associated theoretical guidance to accommodate time-varying networks. Tsunami increases the pacing gain of the steady phase to 1.5, which senses network changes quickly but inevitably causes a buildup of bottleneck buffer packets. PCC is unsuitable for a time-varying environment due to the long time to converge to its target. For BBR v2, CUBIC and BBR-ACD are sensitive to packet loss, and inevitable loss in a time-varying network environment can lead to throughput degradation. Moreover, TCP D* obtains the lowest latency among these CCAs. The main idea behind TCP D* is to use cwnd instead of pacing rate to implement the mechanism

[4]https://github.com/cx9208/bbrplus



(a) 4G trace (b) 5G trace (c) 24Mbps trace

(d) T-Mobile with subway in C2TCP (e) T-Mobile with times in C2TCP (f) Wifi

Fig. 7. Comparing BBR-FIT with baseline algorithms through the throughput and latency across different networks.

of BBR. Therefore, TCP D* is more moderate than BBR. Although TCP D* improves the delay performance, BBR-FIT is about 4× better than its throughput performance while increasing latency by about 1.1×. Although SBBR effectively reduces latency and improves throughput compared to BBR, BBR-FIT still outperforms it by about 3× in throughput performance and 2× in latency performance. Meanwhile, BBR-FIT automatically detects the bandwidth with a pacing gain close to 1 for low latency in wired networks.

C. TCP Fairness

BBR-FIT likely coexists with other TCP flows in the bottleneck link. a scheme that is too aggressive may starve others, while a scheme that is too conservative may starve for a long time. Neither of them is a good candidate. We have proven that BBR-FIT performs very well in different environments. Does it get better performance by adding aggressiveness? We will evaluate BBR-FIT's TCP friendliness.

Specifically, a Sender sends a CUBIC flow to the Receiver. We chose CUBIC because it is the default TCP in Linux operating systems. At the same time, we send another flow from another sender to the client. Each test is run three times, and the throughput of both flows in the evaluation is recorded. Since the performance of CUBIC increases as the buffer size increases, we vary the switch buffer size from 1BDP to 2BDP for a fair comparison.

Fig. 8 shows the throughput ratio of different CCAs to CUBIC under different buffer sizes. The closer the throughput ratio to 1, the better the CCA's friendliness. Compared to the well-performing CCAs in Table III, such as Tsunami, PCC, and Stateful-BBR, BBR-FIT is more friendly. The BBR-FIT is designed with the network congestion degree in mind. The results demonstrate that BBR-FIT gets better performance not by adding aggressiveness to penalize other TCP flows.

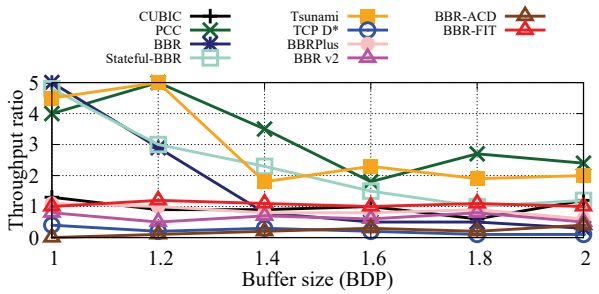


Fig. 8. Friendliness Ratio of different CCAs across different RTTs.

Moreover, we investigate the behavior of BBR-FIT in the presence of other BBR-FIT flows. We choose one of the Senders to send the traffic to the Reciever. Every 50s, we add another flow to the network destined for the same Reciever and record the average throughput of each flow. Each evaluation is completed in the 300s. The results are shown in Fig. 9. both BBR-FIT and BBR achieve appropriate fairness. The results illustrate that in the presence of other BBR-FIT flows, the bandwidth will be shared fairly among the competing flows.

D. The Impact Of RTT And Buffer Size

Deep buffer is the characteristic of the cellular network, leading to the bufferbloat problem [24]. BBR-FIT is designed to obtain both high throughput and low latency. Do different RTT and buffer sizes have a big impact on BBR-FIT? To answer this question, we change RTT and buffer size of the simulation setting to see their impact on the performance

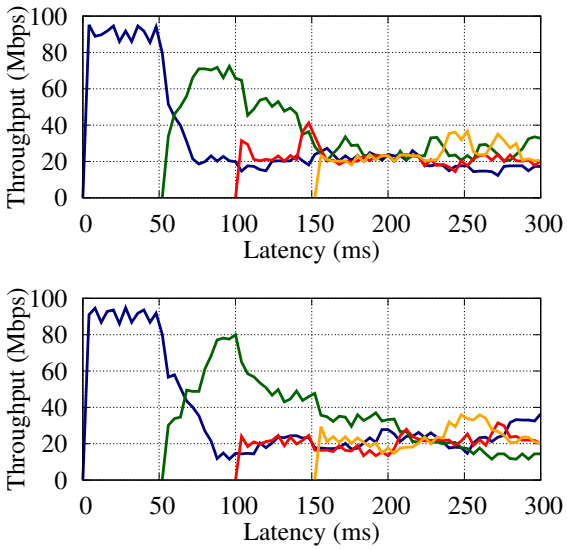


Fig. 9. Throughput dynamics of different flows competing on an Internet link for BBR and BBR-FIT.

of BBR-FIT. To make the results in each scenario more convincing, we design the result as the ratio of the score obtained by BBR-FIT to the score obtained by a CUBIC flow in the same scenario. Therein, the score is calculated by $\frac{Throughput}{Delay}$, which is presented in [23].

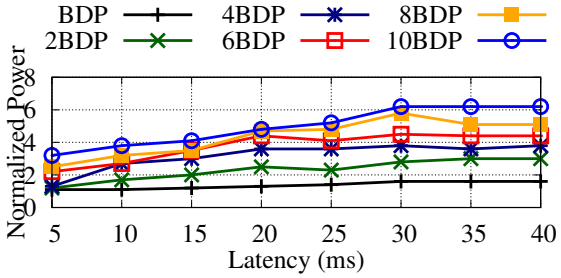


Fig. 10. BBR-FIT's Normalized Power (Norm. to Cubic) across RTTs and buffer sizes

Results are shown in Fig. 10, BBR-FIT can obtain a minimum of 1.2× higher score than CUBIC and a maximum of 6.2× higher score than CUBIC. In addition, the score of CUBIC decreases due to its increased latency in the deep buffer, while BBR-FIT maintains good performance, resulting in an increase in the relative performance of BBR with respect to CUBIC. This demonstrates the role of the reward function (Equation (3)) in BBR-FIT, enabling BBR-FIT to take into account both throughput, latency, and retransmission rate.

V. Linux Implementation

In this section, we evaluate the performance of BBR-FIT in the wireless network in Ningbo City. As shown in Fig. 11, the laptop and ALiCloud server are connected through a router (Xiaomi A100). In the evaluation, the laptop is responsible for sending data. Considering baseline algorithms in Section IV, we use three performance metrics to evaluate

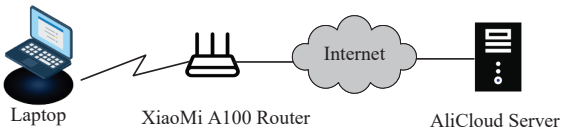


Fig. 11. The topology used for Section V.

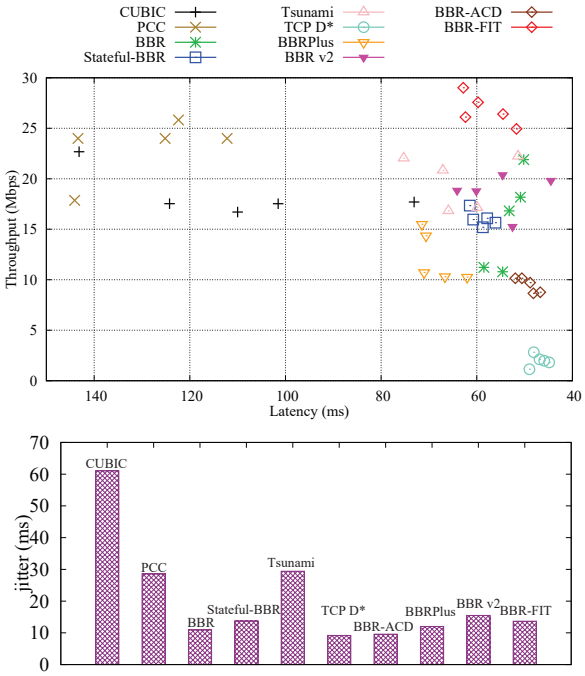


Fig. 12. Average throughput and overall averaged jitter.

their performance: delay, jitter, and throughput. For reliability, we perform each algorithm five times, all at the same time and at a fixed location. Fig. 12 shows the overall averaged throughput, delay, and jitter (defined as the same as [22]).

As expected, BBR-FIT obtains the maximum averaged throughput, about 2× that of BBR. Also, the jitter and delay of BBR-FIT are 1.1× and 1.2× higher than those of BBR, respectively. The result demonstrates that BBR-FIT, instead of simply increasing the aggressiveness to get higher throughput like Tsunami.

VI. Related Work

We present congestion control mechanisms from two aspects: Traditional CCAs and Deep RL-based CCAs.

A. Traditional CCAs

Recently, Many CCAs have been developed to laser-focus on a specific network. For instance, STCC targets data center networks while TCP-WBQ [25] works for cellular networks. Many computing resources and services are deployed on datacenter networks (DCNs) to provide a high QoE. Traditional CCAs do not consider the network heterogeneity between data

centers [26], resulting in severe network performance degradation. STCC adaptively calculates optimal CCA parameters for each flow by periodically monitoring network information through the software-defined networking.

Cellular networks have highly volatile channels, rapidly fluctuating bandwidth, random packet loss, etc. These unique characteristics make it challenging for TCP to achieve low latency and high throughput in cellular networks. Traditional TCP variants perform poorly in cellular networks. TCP-WBQ attributes the problem to the inability to estimate the degree of congestion for fine-grained congestion control. To address this problem, TCP-WBQ constructs a backlog queueing model to define congestion and random packet loss, thus maintaining a tradeoff between high throughput and congestion avoidance. A particular case among these algorithms is BBR. Google tried to extend it to a generic CCA for all network scenarios.

B. Deep reinforcement learning based CCAs

A CCA can be summarized as mapping the sender-aware network information into predefined actions. The performance degrades significantly when the sender-perceived network state differs significantly from the actual network state. Compared to learning-based CCA, manual tuned CCA adapts to new environments and requires a lot of pre-experimentation.

Deep RL has been applied to network systems. The strength of RL lies in its ability to make decisions based on experience without manually designing strategies. To address the problem of slow adaptation for TCP to network changes, [27] proposes a solution. It use network information from endpoints in a software-defined environment to infer congestion. However, as its authors acknowledge, it requires extended analysis and manipulation to improve the understanding of the environment. To resolve the above problem, TCP-NeuRoc [9] combines the Deep Deterministic Policy Gradient with a probabilistic exploration framework. TCP-NeuRoc achieve the best throughput-latency tradeoff. Unlike others, instead of requiring an exact network model and training data, it learns from experience using measurement features to determine how to adjust the congestion window size.

VII. CONCLUTION

In this paper, we address the low performance of the BBR algorithm in time-varying networks and propose a learning-based CCA, called BBR-FIT. By combining the congestion control strategy of BBR and the wisdom of RL techniques, the sensitivity of BBR-FIT to bandwidth changes is improved. Extensive testbed experiments and MahiMahi simulations demonstrate that compare to BBR, BBR-FIT is adaptive and improves link utilization by $2\times$ without increasing the latency. We conclude that BBR-FIT combined with RL can adapt to time-varying network scenarios.

REFERENCES

[1] M. Uitto, A. Heikkinen. "Evaluation of live video streaming performance for low latency use cases in 5G", In Proc: EuCNC/6G Summit, 2021, pp. 431-436.

[2] Cisco Annual Internet Report (2018-2023) White Paper, 2020, [online] Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[3] J. Lorincz, Z. Klarin, J. Ožegović. "A comprehensive overview of TCP congestion control in 5G networks: research challenges and future perspectives". Sensors. vol. 21, pp. 4510-4557,June. 2021.

[4] Y. Ren, W. Yang, X. Zhou, et al. "A survey on TCP over mmWave". Computer Communications, vol. 171, pp. 80-88, Jan. 2021.

[5] M. Alipio, N. M. Tiglao, F. Bokhari, et al. "TCP incast solutions in data center networks: a classification and survey". Journal of Network and Computer Applications, vol.146, pp. 102421-102431, Nov. 2019.

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

[7] Y. Cao, A. Jain, K. Sharma, et al. "When to use and when not to use BBR: An empirical analysis and evaluation study", In Proc: Internet Measurement Conference, 2019, pp. 130-136.

[8] M. Mezzavilla et al., "End-to-End simulation of 5G mmWave networks," in IEEE Communications Surveys and Tutorials, vol. 20, pp. 2237-2263, Nov. 2018.

[9] J. R. S. Veluswami, K. Chinnusamy, K. Kumar, et al. "Improvement of transmission control protocol for high bandwidth applications", Wireless Personal Communications, vol. 117, no. 4, pp. 3359-3379, Jan. 2021.

[10] W. Li, S. Gao, X. Li, Y. Xu and S. Lu, "TCP-NeuRoc: neural adaptive TCP congestion control with online changepoint detection," in IEEE Journal on Selected Areas in Communications, vol. 39, pp. 2461-2475, Aug. 2021

[11] F. Y. Yan et al., "Pantheon: the training ground for internet congestion-control research", In Proc: USENIX, 2018, pp. 731-743.

[12] T. Flach et al., "Reducing web latency: the virtue of gentle aggression", ACM SIGCOMM Comput. Commun. vol. 43, pp. 159-170, Aug. 2013.

[13] M. Dong et al., "PCC vivace: online-learning congestion control", In Proc: USENIX, 2018, pp. 343-356.

[14] S. Zhang, An evaluation of BBR and its variants, in: Proceedings of ACM Conference, 2017.

[15] J. Schulman, F. Wolski, P. Dhariwal, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347, 2017.

[16] M. Dong, Q. Li, D. Zarchy, et al. "PCC: re-architecting congestion control for consistent high performance". In Proc: USENIX, 2015, pp. 395-408.

[17] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Jacobson, Bbr v2 a model-based congestion control, 2019.URL https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00

[18] L. Guo, Y. Liu, W. Yang, et al. "Stateful-BBR–An enhanced TCP for emerging high-bandwidth mobile networks", In Proc: 2021 IEEE ACM 29th International Symposium on Quality of Service, 2021, pp. 1-9.

[19] T. Lynn, D. Ghosal, "TCP D*: A low latency first congestion control algorithm", arXiv preprint arXiv, vol. 2012, no. 14996, Dec. 2020.

[20] I. Mahmud, G. H. Kim, T. Lubna, et al. "BBR-ACD: BBR with advanced congestion detection", Electronics, vol. 9, no. 1, pp. 136-145, Jan. 2020.

[21] Z. Xia, J. Wu, L. Wu, et al. "RLCC: practical learning-based congestion control for the internet". In Proc: IJCNN, 2021, pp. 1-8.

[22] S. Abbasloo, Y. Xu, H. J. Chao, "C2TCP: a flexible cellular TCP to meet stringent delay requirements". IEEE Journal on Selected Areas in Communications, vol. 43, pp. 918-932, Apr. 2019.

[23] Alfred Giessler, J Haenle, Andreas Konig, and E Pade. Free buffer allocation—An investigation by simulation. Computer Networks, 1978.

[24] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the Internet", Queue, vol. 9, no. 11, pp. 40, Nov. 2011.

[25] J. Tang, Y. Jiang, X. Dai, et al. "TCP-WBQ: a backlog-queue-based congestion control mechanism for heterogeneous wireless networks". Scientific Reports, vol. 12, no. 1, pp. 1-17, Mar. 2022.

[26] Y. Chen, R. Griffith, J. Liu, et al. "Understanding TCP incast throughput collapse in datacenter networks". In Proc: ACM workshop on Research on enterprise networking, pp. 73-82, 2009.

[27] A. Hassan, S. S. Heydari, "TCP congestion avoidance in data centres using reinforcement learning". In Proc: ICACT, 2021, pp. 306-311.

[28] K. Xiao, S. Mao, J. K. Tugnait, "TCP-Drinc: Smart congestion control based on deep reinforcement learning". IEEE Access, vol. 7, pp. 11892-11904, Jan. 2019.