entropy

MDPI

Article

PBQ-Enhanced QUIC: QUIC with Deep Reinforcement Learning Congestion Control Mechanism

Zhifei Zhang [1,2,3,*], Shuo Li [1,2,3], Yiyang Ge [1,2,3], Ge Xiong [2,4], Yu Zhang [5] and Ke Xiong [1,2,3,*]

[1] Engineering Research Center of Network Management Technology for High Speed Railway of Ministry of Education, School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China
[2] Collaborative Innovation Center of Railway Traffic Safety, Beijing Jiaotong University, Beijing 100044, China
[3] National Engineering Research Center of Advanced Network Technologies, Beijing Jiaotong University, Beijing 100044, China
[4] China Software and Technical Service Co., Ltd., Beijing 100081, China
[5] Institute of Economics and Energy Supply and Demand, State Grid Energy Research Institute Co., Ltd., Beijing 102209, China
* Correspondence: zhfzhang@bjtu.edu.cn (Z.Z.); kxiong@bjtu.edu.cn (K.X.)

Abstract: Currently, the most widely used protocol for the transportation layer of computer networks for reliable transportation is the Transmission Control Protocol (TCP). However, TCP has some problems such as high handshake delay, head-of-line (HOL) blocking, and so on. To solve these problems, Google proposed the Quick User Datagram Protocol Internet Connection (QUIC) protocol, which supports 0-1 round-trip time (RTT) handshake, a congestion control algorithm configuration in user mode. So far, the QUIC protocol has been integrated with traditional congestion control algorithms, which are not efficient in numerous scenarios. To solve this problem, we propose an efficient congestion control mechanism on the basis of deep reinforcement learning (DRL), i.e., proximal bandwidth-delay quick optimization (PBQ) for QUIC, which combines traditional bottleneck bandwidth and round-trip propagation time (BBR) with proximal policy optimization (PPO). In PBQ, the PPO agent outputs the congestion window (CWnd) and improves itself according to network state, and the BBR specifies the pacing rate of the client. Then, we apply the presented PBQ to QUIC and form a new version of QUIC, i.e., PBQ-enhanced QUIC. The experimental results show that the proposed PBQ-enhanced QUIC achieves much better performance in both throughput and RTT than existing popular versions of QUIC, such as QUIC with Cubic and QUIC with BBR.

Keywords: QUIC; congestion control; deep reinforcement learning; BBR

Citation: Zhang, Z.; Li, S.; Ge, Y.; Xiong, G.; Zhang, Y.; Xiong, K. PBQ-Enhanced QUIC: QUIC with Deep Reinforcement Learning Congestion Control Mechanism. *Entropy* **2022**, *25*, 294. https://doi.org/10.3390/e25020294

Academic Editor: Donald J. Jacobs

Received: 19 December 2022
Revised: 23 January 2023
Accepted: 2 February 2023
Published: 4 February 2023

1. Introduction

At present, computer networks remain the essential platform for information interaction, where the transport layer plays an influential role. The emergence of modern applications, such as video live broadcast and Internet of Things (IoT), has imposed higher demands on throughput, packet loss rate, and network delay. The development of wireless transmission technologies such as 5G and WiFi has made the network environment even more complex. The Transmission Control Protocol (TCP), as a widely used protocol, experiences problems such as large handshake delay, head-of-line (HOL) blocking, and protocol solidification, which increasingly affect network performance. Compared with TCP, the User Datagram Protocol (UDP) is more efficient for real-time transmission but lacks reliability.

In 2012, Google proposed the Quick UDP Internet Connection (QUIC) protocol [1], which realizes orderly, quick, and reliable transport services in user mode based on UDP. The QUIC protocol reduces the handshake latency to zero round-trip time (RTT) by caching ServerConfig. Moreover, the QUIC protocol natively supports multiplexing techniques

where streams on the same connection do not influence each other, which solves the HOL blocking problem. Additionally, the QUIC protocol decouples the congestion control algorithm from the protocol stack, which is more flexible than TCP. Currently, the QUIC protocol has become one focus of researchers in both academia and industry.

For both TCP and QUIC protocols, congestion control is one of the core mechanisms. The basic logic of TCP congestion control contains congestion sensing and the corresponding disposal pattern, which improves the network utilization by estimating the bandwidth-delay product (BDP) of the link and adjusting the send rate of clients. Since the first congestion control algorithm, Tahoe [2], was proposed, there have been dozens of congestion control algorithms developed that are suitable for different scenarios. The operation rules of traditional congestion control algorithms are shown in Figure 1. Congestion control algorithms work on the sender, sensing the network state and changing the congestion window or transmission rate. Currently widely used control algorithms are heuristic and cannot be optimally executed in dynamic and changeable network environments.

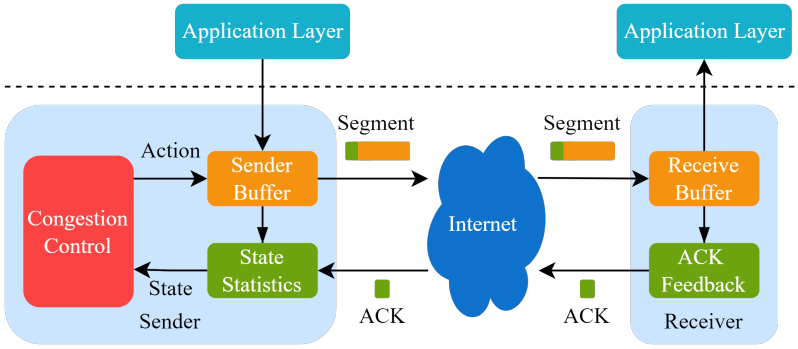


Figure 1. A schematic of congestion control in TCP.

Recently, researchers have started to design TCP congestion control algorithms using machine learning algorithms. There are numerous versions of TCP based on supervised learning, such as TCP with DeePCCI [3] and TCP with LSTM-PTI [4]. Naturally, supervised-learning-based algorithms are only used for passive congestion identification, and training them requires a great deal of labeled data. As the network environment changes, so do the network characteristics. In this case, the supervised learning congestion control algorithm is not suitable for the changing network environment and may no longer be effective. In light of the advantages of the reinforcement learning (RL) method in sequence decision [5,6], researchers all over the world have been trying to apply it to congestion control. So far, some well-known congestion control methods, such as Remy [7], performance-oriented congestion control (PCC)-Vivace [8], Q-learning TCP (QTCP) [9], Orca [10] and Aurora [11] algorithms have been proposed for the TCP protocol. For the QUIC protocol, researchers worldwide have proposed some modified versions of QUIC, i.e., QUIC-go [12], Microsoft QUIC (MsQUIC) [13], Modified QUIC [14], and Quiche [15]. QUIC-go is an implementation of the QUIC protocol in Go. It keeps up to date with the latest request for comments (RFC) and is easy to deploy. MsQUIC is a Microsoft implementation of the QUIC protocol. It optimizes the maximal throughput and minimal latency. Modified QUIC proposes a modification to the handshaking mechanism to minimize the time required to update the CWnd, which results in a smooth variation in the CWnd. This makes modified QUIC protocol easy to deploy and achieves better performance in terms of throughput. Quiche is an implementation of the QUIC protocol in Rust and C. It performs well across different applications, such nginx and curl.

However, for these popular versions of QUIC, only heuristic congestion control algorithms are applied. Most heuristic congestion control algorithms commonly perceive the network state based on simple network models and adopt a fixed strategy, i.e., shrinking the window when packet loss occurs or RTT increases, and augmenting the window when an acknowledge character (ACK) is received. Such a simple strategy makes heuristic con-

gestion control algorithms unable to achieve optimal results in dynamic and changeable network environments. For example, in the WiFi scenario, where obstacles and human activity affect network quality, packet loss events do not imply the occurrence of congestion. However, heuristic congestion control algorithms cannot distinguish that. This issue leads to poor performance of the QUIC protocol in terms of throughput and latency when the network settings change. Therefore, to enable the QUIC to achieve better performance in different network environments, we tried to design DRL-enhanced QUIC using the merits of DRL in environment awareness and decision making.

The contributions of this work can be summarized as follows:

- First, we developed a novel congestion control mechanism, referred to as proximal bandwidth-delay quick optimization (PBQ) by combining proximal policy optimization (PPO) [16] with traditional BBR [17]. It is able to effectively improve the convergence speed and link stability during the training phase. We then applied the presented PBQ to the QUIC protocol and formed a new version of QUIC, i.e., PBQ-enhanced QUIC, which aims to enhance its adaptivity and throughput performance.
- Second, for the purpose of reducing the action space and establishing the connection of the values of the congestion window (CWnd) for each interaction, we used continuous action ratio as the output action of PBQ's agent. Additionally, in the design of the utility function, we used a relatively simple formulation of the objective function as the optimization objective and introduced a delay constraint. By doing so, our proposed PBQ-enhanced QUIC achieves higher throughput and maintains a low RTT.
- Third, we built a reinforcement learning environment for QUIC on the network simulation software ns-3, where we trained and tested PBQ-enhanced QUIC. The experimental results showed that our presented PBQ-enhanced QUIC achieves much better RTT and throughput performance than existing popular versions of QUIC, such as QUIC with BBR and QUIC with Cubic [18].

The organization of the rest of this paper is as follows: Section 2 describes the QUIC protocol, including the handshake phase, multiplexing, connection migration, and congestion control. Section 3 gives the design of PBQ and PBQ-enhanced QUIC. Section 4 introduces the training link and testing link and evaluates the performance of PBQ-enhanced QUIC compared with that of QUIC using different congestion control algorithms. Finally, Section 5 gives the conclusions.

2. QUIC Protocol

2.1. The Basic Concept of QUIC

The QUIC protocol is a secure and reliable data transfer protocol first proposed by Google in 2012, which was released as a standardized version of QUIC, RFC 9000 [19], by the Internet Engineering Task Force (IETF). The bottom layer of QUIC is UDP, which makes it compatible with current network protocols. At the same time, the QUIC protocol is compatible with traditional congestion control algorithms in TCP. The QUIC protocol has numerous improvements over TCP, mainly in handshaking, multiplexing, and connection migration.

2.2. Handshake

TCP is a plaintext transport protocol and transport layer security (TLS) is required to implement data encryption transmission. In TLS 1.2, two RTTs are required for the TLS handshake phase and three RTTs for the total handshake delay. Under TLS 1.3, the TLS handshake phase requires one RTT, and the handshake phase still requires two RTTs.

Comparatively, the QUIC protocol optimizes the handshake process. In the handshake phase, QUIC incorporates the transmission parameters into the encryption part. When the first connection is established, the client sends authentication- and encryption-related information to the server. It takes only one RTT to establish a connection. Alternatively, we can use quantum key distribution to replace the original encryption information

exchange [20]. When the connection is established once again, the client uses the preshared key to establish an encrypted connection with the server during zero RTT.

2.3. Multiplexing

Multiplexing is multiple streams on a connection at the same time, commonly on a Hyper Text Transfer Protocol (HTTP) stream. For HTTP/2, a TCP connection can support multiple HTTP streams. However, HOL blocking in the TCP protocol causes interference between HTTP streams, which affects their performance. In advance, multiple streams can be created on a single QUIC connection, which reduces the handshake frequency. The QUIC protocol is implemented based on UDP. Streams on the same QUIC connection are independent of each other, which solves the HOL blocking problem.

2.4. Connection Migration

One TCP connection is known to be identified by a quintuple, i.e., <IP_{source}, $PORT_{source}$, IP_{dest}, $PORT_{dest}$>, where IP_{source} denotes the source IP address, $PORT_{source}$ denotes the source port, IP_{dest} denotes the destination IP address, and $PORT_{dest}$ denotes the destination port [21]. If one term in the quadruple changes, the connection is broken. By using a 64-bit random ID as the connection identifier, the QUIC protocol avoids the effect of network switching. The simple process of connection migration is shown in Figure 2.

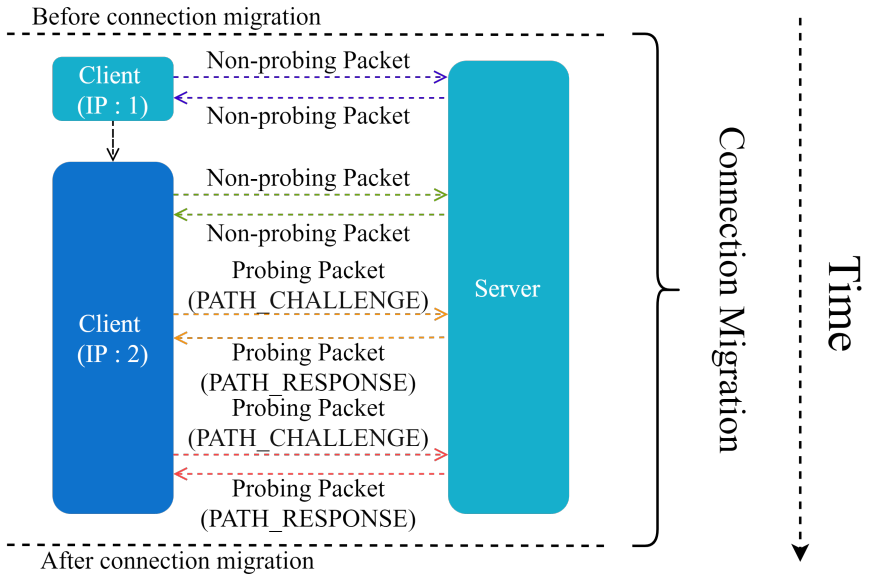


Figure 2. Connection migration.

Particularly, before the client's IP changes, the endpoints communicate via a nondetection packet. After IP changes, the path detection is performed to verify the reachability before connection migration starts. If the path detection fails, the connection migration cannot be performed; otherwise, a fresh connection is established. The IP address is verified between the client and the server, and the endpoint holding the latest IP address of the peer migrates. When migration occurs, the congestion control part and the reliable transport protocol estimation part of the path need to be reset. After the connection migrates, it sends a nonprobe packet.

2.5. Congestion Control in QUIC

The QUIC protocol is a reliable UDP-based data transfer protocol, which makes it compatible with existing network protocols. The congestion control algorithm in TCP is implemented in kernel mode, and if an upgrade is performed, the kernel needs to be recompiled. However, the QUIC protocol straddles kernel mode and user mode, and its congestion control part is in user mode. Thus, it can be easily upgraded. In particular, the

QUIC protocol supports the configuration of different congestion control algorithms for applications, making it possible to optimize for specific applications.

3. The PBQ-QUIC

In this section, we describe PBQ and its application to QUIC to form the PBQ-enhanced QUIC.

3.1. The Basic Idea of PBQ

For clarity, Figure 3 shows the workflow of PBQ. PBQ combines PPO with BBR to accurately identify the network state. It can effectively identify congestion and packet loss events. PBQ is divided into three main parts: Monitoring, Decision, and Pacer modules. The Monitoring module collects environment state and sends it to the Decision module. The Decision module outputs actions, including a_t and the pacing rate, according to the network state. The Pacer module distributes actions to the corresponding senders. In the Decision module, the Controller distributes the network state to the PPO and the BBR. The PPO part outputs a_t according to s_t using the new policy and sends it to the environment through Pacer. The replay memory stores the experience of past interactions. After multiple interactions, the policy networks and the value network are updated using the past interaction experience. Inspired by Orca, our Decision module adopts a two-level regulation mechanism, which is shown in Figure 4. The underlying BBR algorithm performs a classical decision-making behavior driven by ACK. The DRL agent evaluates the network congestion and predicts the BDP according to the state output by the Monitoring module.

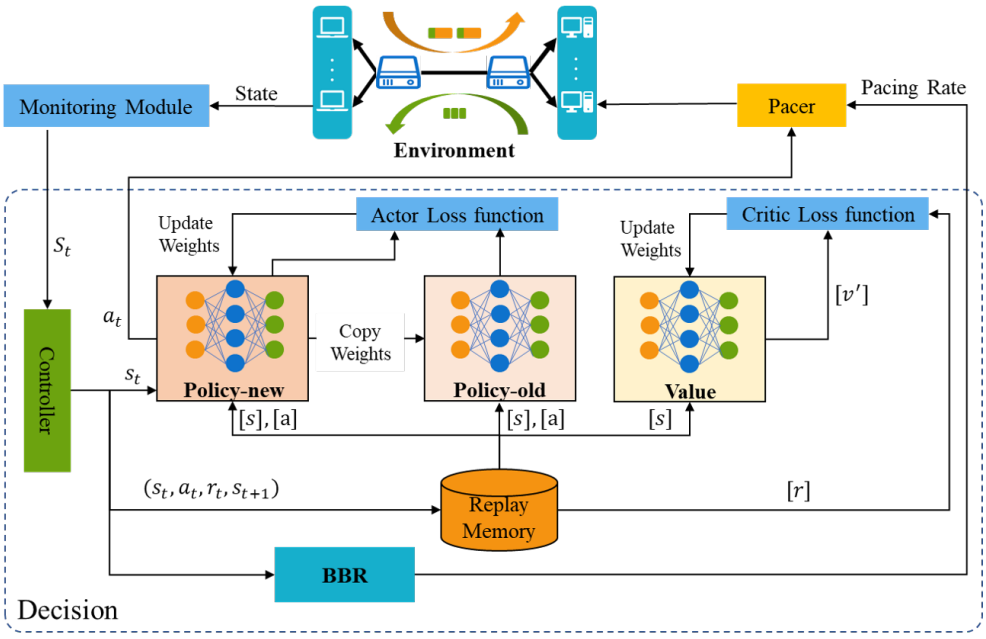


Figure 3. The framework of PBQ.

- Monitoring Module

We set the network state collection interval to 100 ms, and the designed state quantity is shown in Table 1. Because the interval is not strictly equal to 100 ms, we also count the intervals. The remaining states are the statistics within each interval.

Table 1. State statistics description.

State	Description
$CWnd_t$	Current congestion window
$interval_t$	Cata update interval
$deliveryRate_t$	Average delivering rate (throughput)
RTT_t	Averaged RTT
$packetLoss_t$	Average loss rate of packets

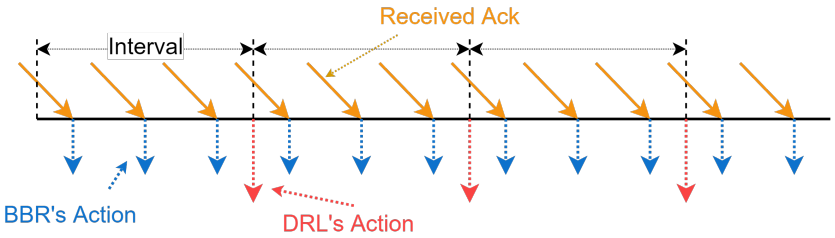


Figure 4. The two-level control logic of PBQ's enhanced QUIC.

- Reward Function Design

We trained PBQ using a linear function with constraints. First, we define a linear utility function as $\alpha * deliveryRate_t - \beta * packetLoss_t$, where α denotes the coefficient of $deliveryRate_t$, $deliveryRate_t$ denotes the delivery rate in time t, β denotes the coefficient of $packetLoss_t$, and $packetLoss_t$ denotes the packet loss rate in time t. Then, we formulate an optimization problem to maximize the linear utility function under a given RTT constraint, i.e., $RTT_t \leq minRTT_t$,$t = 1, 2, 3 \ldots, n$, where RTT_t denotes the last RTT in time t, and $minRTT_t$ denotes the minimum RTT from the establishment of the connection to time t.

$$\begin{aligned} \max \quad & \alpha * deliveryRate_t - \beta * packetLoss_t \\ s.t. \quad & RTT_t \leq minRTT_t \text{ ,} t = 1, 2, 3 \ldots, n \quad . \end{aligned} \tag{1}$$

We use the maximization objective as the base reward function:

$$R_t = \alpha * deliveryRate_t - \beta * packetLoss_t \tag{2}$$

With the delay constraint, the final reward function is

$$\begin{aligned} R_t &= \alpha * deliveryRate_t - \beta * packetLoss_t \\ R_t &= \begin{cases} R_t & RTT_t \leq \gamma * minRTT_t \\ R_t - \eta & RTT_t > \gamma * minRTT_t \end{cases} \end{aligned} \tag{3}$$

where γ denotes the penalty threshold, and η denotes the penalty term when $RTT_t > \gamma * minRTT_t$.

- Action Design

Reinforcement learning action types can be divided into two categories, discrete and continuous actions. We first used discrete actions, but analyzing the experimental results, we found that when using discrete actions, the output of the agent considerably fluctuated, and it was difficult to achieve better performance in the initial stages of the interaction. In general, the discrete action does not favor the early stability of the network links. The final solution in this study draws on the additive increase multiplicative decrease (AIMD) [22] idea in traditional congestion control, and we set the action output as $CWndRatio$. The mapping relationship between $CWndRatio$ and the congestion window is as follows:

$$newCWnd = CWnd * 2^{CWndRatio} \tag{4}$$

The relationship between the values of CWnd for each interaction was established, which reduces the fluctuation of the action and ensures the excellent performance of the model.

- Learning Algorithm for PBQ

For the DRL agent, PPO is employed, which is a classical actor-critic algorithm. We used tanh as the activation function in both actor and critic neural networks. Because the state transitions in traditional congestion algorithms are not complex, we preferred to build a simpler neural network and train on it. Finally, we constructed a three-layer neural network, where the hidden layer contains 64 neurons.

For clarity, the PBQ training phase is summarized in Algorithm 1. On lines 1–2 of Algorithm 1, the training episode $Episodes$ is set and the replay memory D is initialized, which is used to store state, action and reward. The policy parameters θ_{new} and value parameters ϕ are also initialized with random weights. We set policy parameters θ_{old} equal to θ_{new}. Then, on lines 4–16, at each episode, we first reset the environment and obtain the initial state s_0 and initial reward r_0. We use PPO to regularly output α_t according to the new policy $\pi_{\theta_{new}}$. We map α_t to $CWnd_t$, which is also the estimated link BDP. Then, $CWnd_t$ is preformed into environment. In each step, PBQ collects network states s_t and s_{t+1}, action value a_t, and the corresponding reward r_t, and updates the replay memory D. The policy parameters θ_{new} and value functions ϕ are updated when the number of steps accumulates to the policy update threshold $update\_timestep$. The underlying BBR method is driven by ACK to control the specific pacing rate of the clients.

Algorithm 1 PBQ's learning algorithm.

1: Initialize replay memory D and training episode $Episodes$
2: Initialize policy parameters θ_{new}, value parameters ϕ, $\theta_{old} \leftarrow \theta_{new}$
3: **for** $episode$ = 1 to $Episodes$ **do**
4: Reset environment, obtain initial state s_0 and initial reward r_0
5: $i \leftarrow 1$
6: **for** t = 1 to $steps$ **do**
7: **if** time to play PPO action **then**
8: Run policy $\pi_{\theta_{new}}$, obtain PPO action α_t
9: Map α_t to $CWnd_t$, $CWnd_t = CWnd_{t-1} * 2^{\alpha_t}$
10: Perform $CWnd_t$
11: **else**
12: Play a BBR action, perform $pacingRate_t$
13: **end if**
14: Collect $\{s_t, a_t, r_t, s_{t+1}\}$, update D
15: $t = t + 1$
16: **if** t % $update\_timestep == 0$ **then**
17: $\theta_{old} \leftarrow \theta_{new}$, update θ_{new} and ϕ
18: **end if**
19: **end for**
20: **end for**

- Pacer

The Pacer module is responsible for distributing the actions output by the Decision module to the corresponding clients. The action distribution is driven by an update callback for the CWnd and an update callback for the pacing rate. For pacing rate, when the client receives an ACK or packet loss occurs, the underlying BBR adjusts the pacing rate and triggers a pacing rate update callback. For CWnd, the Decision module receives the updated network state from the Monitoring module; the RL part outputs α and calculates the current CWnd; then, the callback function is called to update the CWnd on the client via Pacer module. As shown in Figure 5, the Pacer module distributes actions to the corresponding clients based on their ids.

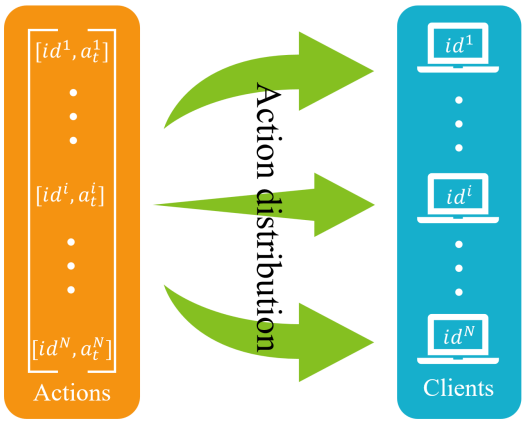


Figure 5. Action distribution in the Pacer module.

3.2. PBQ-Enhanced QUIC

Figure 6 shows the differences in congestion control between PBQ-enhanced QUIC and traditional QUIC. QUIC uses Cubic as the default congestion control algorithm. Cubic is a heuristic algorithm that is driven by events at the sender side, including ACK receipt, packet loss, etc. The policy of Cubic is based on packet loss events, which leads to poor performance in scenarios where packet loss is present. PBQ is a combination of the DRL method PPO and BBR with the advantages of both. Our deployment adopts the client/server (C/S) mode, which only modifies QUIC in client and adopts the standard QUIC implementation in the server. We deeply integrated PBQ into the QUIC client. We added the Monitoring module for network state statistics, the Decision module for rate control, and the Pacer module for rate distribution. The Decision and Pacer modules are asynchronously executed, and they do not affect the behavior of the clients.

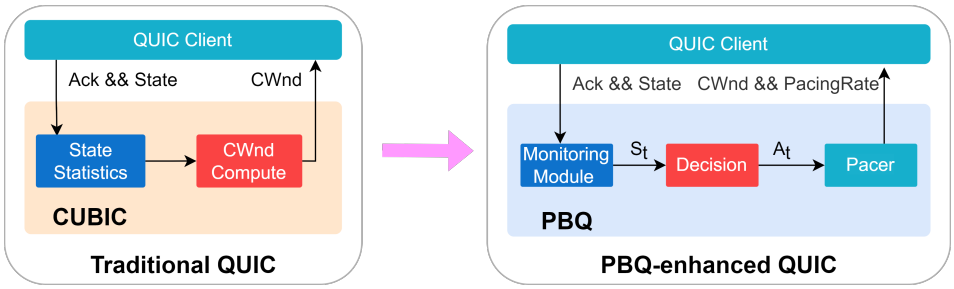


Figure 6. PBQ-enhanced QUIC.

We specifically designed states and actions. The Monitoring module periodically collects the state of the environment and sends the processed data to the Decision module. We collected a number of environment states, including pacing rate, current RTT and minimum RTT, interval, packet loss rate, CWnd, and so on. The output of the Pacer module includes the CWnd and pacing rate of the senders. The BBR regulates the pacing rate of the clients, and the DRL agent gives the congestion window as the predicted BDP. Then, the Pacer module changes the sending behavior of the clients.

4. Simulation Performance

We tested the PBQ-enhanced QUIC in various scenarios on the open-source network simulator ns-3 [23] and compared the results with those of QUIC using different congestion control algorithms. We wrote simulation scripts in C++ and implemented our DRL agent based on MindSpore in python. Then, we ran the comparison experiments on a Dell PowerEdge R840 server with 256 G memory, 64 cores, and a GeForce RTX 3090 GPU. We tested the model on a large number of links with different parameters and analyzed the sensitivity of PBQ-enhanced QUIC to the number of link flows and packet loss rate.

When the testing and training links were quite different, the PBQ-enhanced QUIC still performed well.

4.1. Our Simulation Environment

The application of DRL methods is inseparable from the interaction with the environment. Several researchers have implemented their own simulation environments, but they all target the TCP domain, and there is no mature solution for the simulation environment of the QUIC protocol alone. At the same time, we think that the development of a network simulation environment is a huge and meticulous project that should be focused on the study of congestion control algorithms; therefore, we chose the ns-3 platform to build the training and testing environment. We built our own simulation environment based on NS3-gym [24] and Quic-NS-3 [25] to test PBQ-enhanced QUIC.

4.2. Training

We also applied the PPO method to the congestion control part of QUIC. We trained PPO and PBQ on the link shown in Figure 7. For training, we set N to two, which means that two traffic flows shared a link. The link parameters are shown in Table 2. Our model converged in 50 epochs, each consisting of 1200 steps. In contrast, the experimental results showed that after combining with BBR, the model converged faster, and the throughput and RTT performance were more stable in the initial interaction.

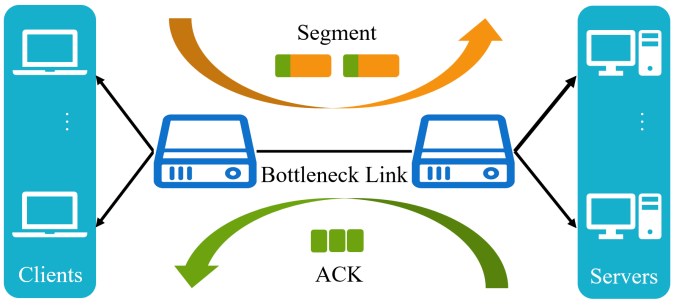
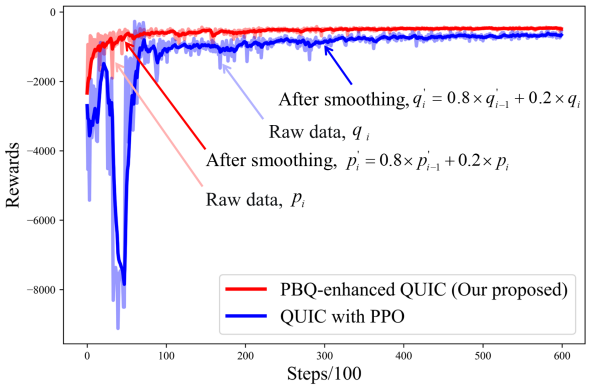


Figure 7. The simulation link of PBQ.

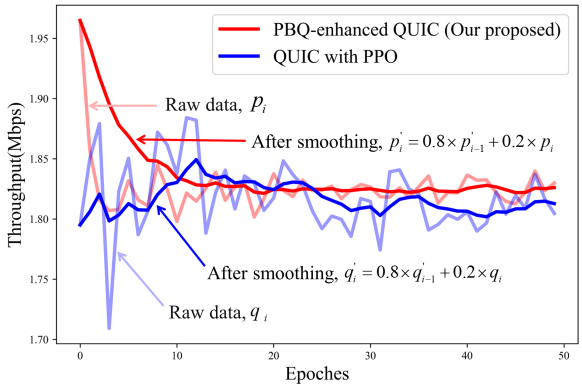
Table 2. Training link.

Attribute	Value
Number of flows	2
Bottleneck bandwidth	2 Mbps
RTT	30 ms
Queue capacity	75 Kilobytes
Queue scheduling algorithm	First Input First Output (FIFO)

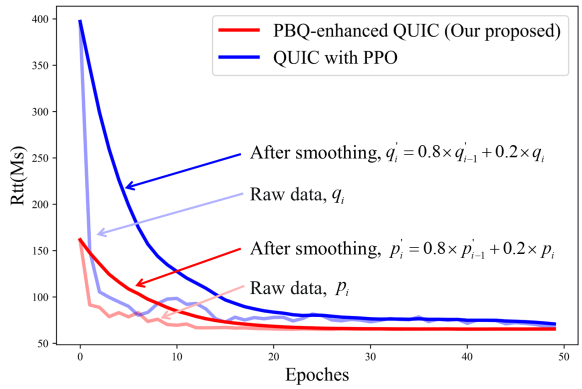
As illustrated in Figure 8b,c, the algorithm we propose can effectively improve the stability of the pretraining action while ensuring the quality of the link. As shown in Figure 8a, the combination with the BBR effectively speeds up the speed of convergence, which solves the problems we described before.



(**a**)



(**b**)



(**c**)

Figure 8. Training process after 600 episodes: (**a**) training reward; (**b**) training throughput; (**c**) training RTT.

4.3. Testing

Similarly, we compared PBQ-enhanced QUIC with QUIC using current learning-based congestion control algorithm Remy and heuristic algorithms such as Bic [26], Cubic, Low Extra Delay Background Transport (LEDBAT) [27], NewReno [28], Vegas [29], and BBR. [id = S.L.] The link parameters are shown in Table 3.

Table 3. Testing link.

Attribute	Value
Number of flows	2∼8
Bottleneck bandwidth	10 Mbps
RTT	100 ms
Packet loss rate	0%∼10%
Queue capacity	75 Kilobytes
Queue scheduling algorithm	FIFO

In traditional algorithms, BBR is based on network modeling, while the remaining algorithms take packet loss events as congestion occurrence signals. In this part, we still used the dumbbell link in Figure 7. To demonstrate the adaptability of our algorithm, there was a large deviation between our test and training links. Specifically, we determined the performance of PBQ-enhanced QUIC in different scenarios, including its excellent adaptation and tolerance to packet loss and number of flows.

4.4. Packet Loss Rate

We tested the proposed algorithm using a previously trained model in a different network scenario. The throughput changes with the packet loss rate as illustrated in Figure 9a, and the RTT is shown in Figure 9b.

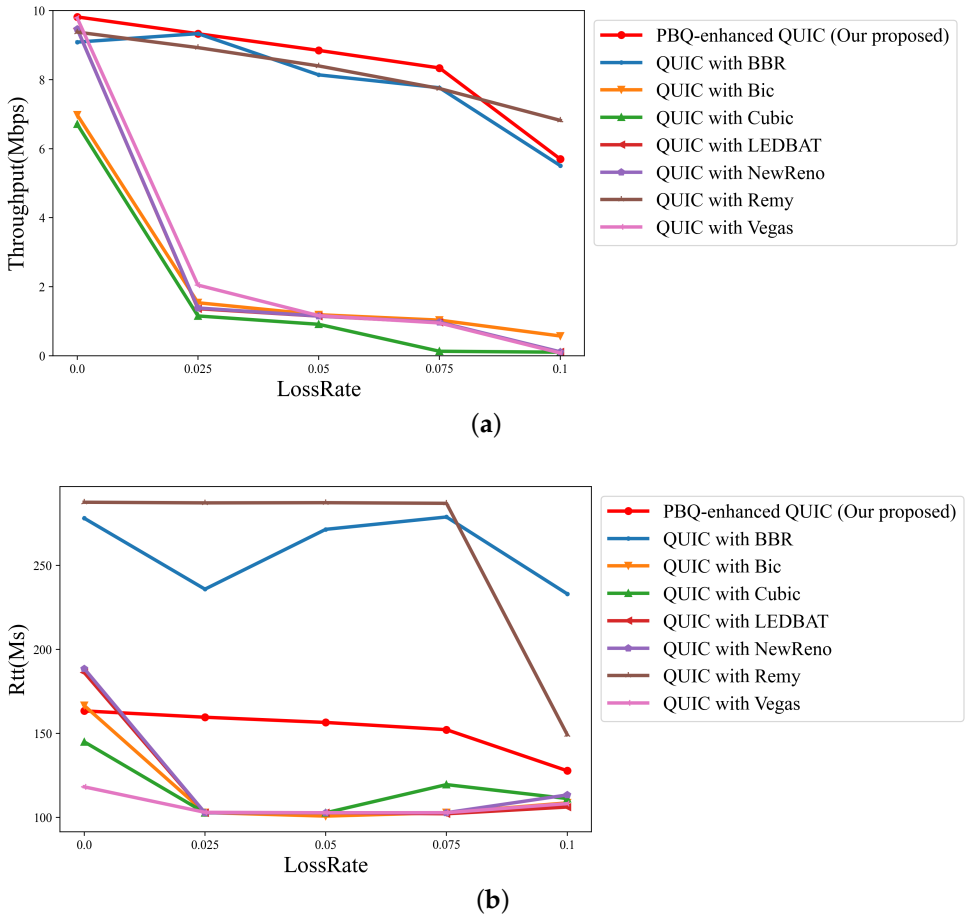


Figure 9. Link quality of schemes across different packet loss rates with flow number 2: (**a**) throughput; (**b**) RTT.

When packet loss occurs, QUIC with congestion control algorithms that take packet loss events as a signal, such as QUIC with Cubic and QUIC with NewReno, frequently enter fast recovery, resulting in low throughput and low RTT. While QUIC with BBR requires constant probing of the network, QUIC with Remy relies on a state-action table. It is difficult for them to balance throughput and RTT. Our algorithm can efficiently identify the overall state of the network and can output relevant actions by periodically collecting the network state. As the packet loss rate increases, the throughput performance of our algorithm is less affected, and it is the best among the different packet loss rates. Furthermore, our proposed algorithm achieves higher throughput analso has better packet loss tolerance ability than QUIC with Remy and QUIC with BBR.

4.5. Flow Number

Then, the packet loss rate was set to 2.5%, and we modified the number of flows and compared the throughput and delay performance of different QUIC implementations. The results are shown as Figure 10a,b. Figure 10a represents the total link throughput. Figure 10b indicates the average RTT.

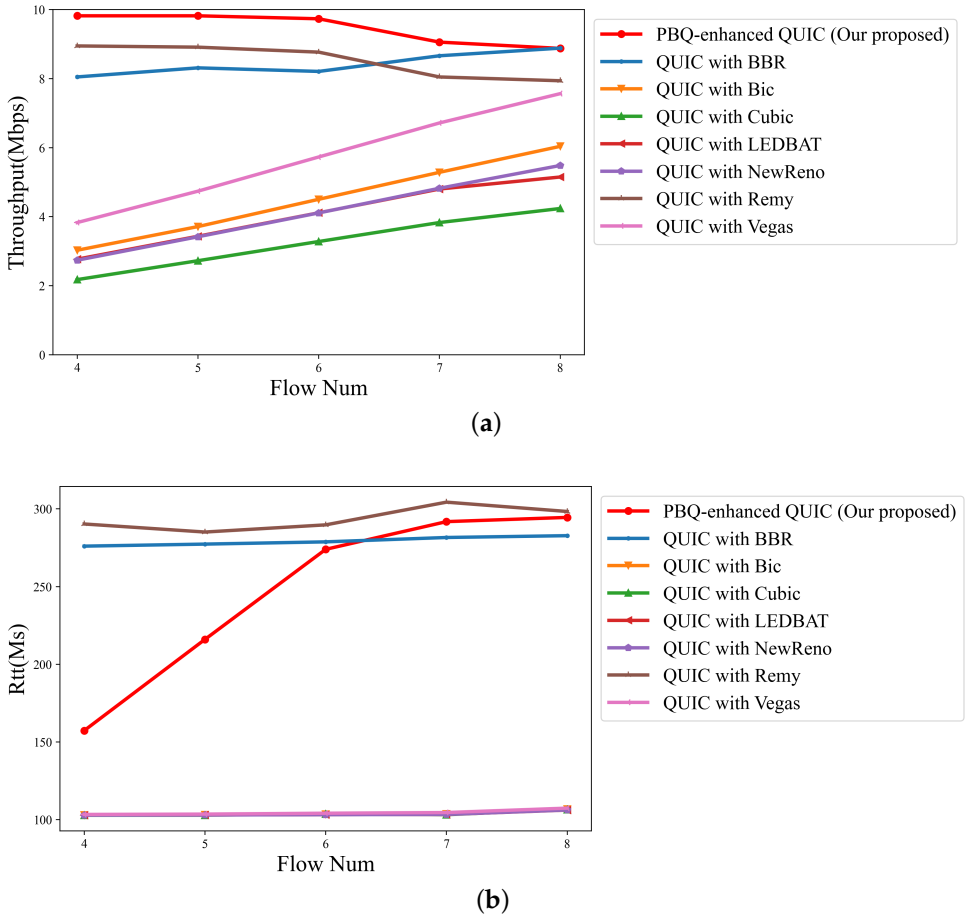


Figure 10. Link quality of schemes for different flow numbers with packet loss rate of 2.5%: (**a**) throughput; (**b**) RTT.

Owing to the presence of random packet loss, algorithms that take packet loss events as congestion occurrence signals fail to accurately identify the cause of congestion. In the policies of these congestion control methods, the current link can only support a low throughput; thus, the throughput is at a low level. In this case, the flows barely affect each other. As a result, as shown in Figure 10a, the total throughput of the link linearly increases with the number of flows. They cannot accurately estimate the link BDP and occupy the

full link bandwidth, so the average link RTT is close to the default RTT. Compared with QUIC with BBR and QUIC with Remy, PBQ-enhanced QUIC reduces the detection process of link BDP and achieves excellent throughput and RTT by optimizing the reward function.

5. Conclusions

In this study, we first developed a two-level regulatory mechanism PBQ, which combines the heuristic algorithms BBR and DRL to approximate PPO. The convergence rate of the model is accelerated by using BBR to estimate the specific transmission rate. Moreover, we proposed PBQ-enhanced QUIC, an implementation of QUIC that uses PBQ as a congestion control algorithm. Unlike QUIC with heuristic congestion control algorithms, our QUIC implementation learns congestion control rules from experience by using RL signals. Therefore, our QUIC implementation can be better adapted to various network settings.

As shown in Section 4.2, the combination with BBR can effectively speed up the convergence of the PPO during the training phase on the premise of ensuring link quality. As shown in Sections 4.2 and 4.4, compared with other QUIC versions, PBQ-enhanced QUIC achieves higher throughput performance in various network settings. PBQ-enhanced also has better RTT performance than QUIC with BBR and QUIC with Remy. We think that combining DRL methods with traditional algorithms to design congestion control mechanisms for QUIC will be a major trend in the future, and PBQ-enhanced QUIC provides a new idea to do so.

In future work, we plan to build a testbed in real-world networks and test PBQ-enhanced QUIC on it. We will improve the PBQ-enhanced QUIC based on its performance in real-world networks. Moreover, different applications have different requirements on network metrics, so we plan to design congestion control algorithms for different applications by taking advantage of the feature that the congestion control part of QUIC is implemented in user mode. We expect different applications to perform efficiently on the same network.

Author Contributions: Conceptualization, Z.Z. and S.L.; methodology, Z.Z. and S.L.; software, S.L. and Y.G.; validation, S.L. and G.X.; formal analysis, Z.Z. and S.L.; investigation, S.L. and Y.Z.; resources, Z.Z. and G.X.; data curation, S.L. and Y.Z.; writing—original draft preparation, S.L.; writing—review and editing, S.L. and K.X.; visualization, S.L. and Y.G.; supervision, Z.Z. and K.X.; project administration, Z.Z; funding acquisition, Z.Z. and K.X. All authors have read and agreed to the published version of the manuscript.

Funding: This study was funded in part by CAAI-Huawei MindSpore Open Fund Grant No. CAAIXSJLJJ-2021-055A and in part by the Fundamental Research Funds for the Central Universities Grant No. 2022JBZY021.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

TCP	Transmission Control Protocol
HOL	Head-of-line
QUIC	Quick UDP Internet Connection
RTT	Round-trip time
DRL	Deep reinforcement learning
PBQ	Proximal bandwidth-delay quick optimization
BBR	Bottleneck bandwidth and round-trip propagation time
PPO	Proximal policy optimization

CWnd	Congestion window
IoT	Internet of Things
UDP	User Datagram Protocol
BDP	Bandwidth-delay product
RL	Reinforcement learning
PCC	Performance-oriented congestion control
QTCP	Q-learning TCP
MsQUIC	Microsoft QUIC
RFC	Request for comments
ACK	Acknowledge character
IETF	Internet Engineering Task Force
TLS	Transport layer security
HTTP	Hyper Text Transfer Protocol
AIMD	Additive increase multiplicative decrease
C/S	Client/server
LEDBAT	Low extra delay background transport

References

1. Langley, A.; Riddoch, A.; Wilk, A.; Vicente, A.; Krasic, C.; Zhang, D.; Yang, F.; Kouranov, F.; Swett, I.; Iyengar, J.; et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In Proceedings of the ACM SIGCOMM, Los Angeles, CA, USA, 21–25 August 2017.
2. Jacobson, V. Congestion avoidance and control. *ACM SIGCOMM Comput. Commun. Rev.* **1988**, *18*, 314–329. [CrossRef]
3. Sander, C.; Rüth, J.; Hohlfeld, O.; Wehrle, K. Deepcci: Deep learning-based passive congestion control identification. In Proceedings of the 2019 Workshop on Network Meets AI & ML, Beijing, China, 23 August 2019; pp. 37–43.
4. Chen, X.; Xu, S.; Chen, X.; Cao, S.; Zhang, S.; Sun, Y. Passive TCP identification for wired and wireless networks: A long-short term memory approach. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 717–722.
5. Brown, N.; Sandholm, T. Safe and nested subgame solving for imperfect-information games. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1–11 .
6. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv* **2017**, arXiv:1712.01815.
7. Winstein, K.; Balakrishnan, H. TCP ex Machina: Computer-Generated Congestion Control. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, Hong Kong, China, 27 August 2013.
8. Dong, M.; Meng, T.; Zarchy, D.; Arslan, E.; Gilad, Y.; Godfrey, P.B.; Schapira, M. PCC Vivace: Online-Learning Congestion Control. In Proceedings of the 15th Usenix Symposium on Networked Systems Design and Implementation (Nsdi'18) 2018, Renton, WA, USA, 9–11 April 2018; pp. 343–356.
9. Li, W.; Zhou, F.; Chowdhury, K.R.; Meleis, W. QTCP: Adaptive Congestion Control with Reinforcement Learning. *IEEE Netw. Sci. Eng.* **2019**, *6*, 445–458. [CrossRef]
10. Abbasloo, S.; Yen, C.-Y.; Chao, H.J. Classic Meets Modern: a Pragmatic Learning-Based Congestion Control for the Internet. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, Online, 10–14 August 2020; pp. 632–647.
11. Jay, N.; Rotman, N.; Godfrey, B.; Schapira, M.; Tamar, A. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019; pp. 3050–3059.
12. Clemente L, Seemann M. Quic-go. Available online: https://github.com/lucas-clemente/quic-go (accessed on 3 August 2016).
13. Microsoft. msquic. Available online: https://github.com/microsoft/msquic (accessed on 26 October 2019).
14. Kharat, P.; Kulkarni, M. Modified QUIC protocol with congestion control for improved network performance. *IET Commun.* **2021**, *15*, 1210–1222. [CrossRef]
15. Cloudflare. Quiche. Available online: https://github.com/cloudflare/quiche (accessed on 1 February 2021).
16. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017** arXiv:1707.06347.
17. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue* **2016**, *14*, 20–53. [CrossRef]
18. Ha, S.; Rhee, I.; Xu, L. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Oper. Syst. Rev.* **2008**, *42*, 64–74. [CrossRef]
19. Iyengar, J.; Thomson, M. *RFC 9000 QUIC: A UDP-Based Multiplexed and Secure Transport. Omtermet Emgomeeromg Task Force*; ACM Digital Library: New York, NY, USA, 2021.
20. Yan, P.; Yu, N. The QQUIC Transport Protocol: Quantum-Assisted UDP Internet Connections. *Entropy* **2022**, *24*, 1488. [CrossRef]
21. Kurose, J.F.; Ross, K.W. *Computer Networking: A Top-Down Approach*, 7th ed.; Pearson FT Press: Upper Saddle River, NJ, USA, 2016.

22. Floyd, S.; Kohler, E.; Padhye, J. *Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)*; RFC: Marina del Rey, CA, USA, 2006; pp. 2070–1721.
23. Henderson, T.R.; Lacage, M.; Riley, G.F.; Dowell, C.; Kopena, J. Network simulations with the ns-3 simulator. *Sigcomm Demonstr.* **2008**, *14*, 527.
24. Gawlowicz, P.; Zubow, A. ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research. In Proceedings of the 22nd International Acm Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Miami Beach, FL, USA, 25–29 November 2019; pp. 113–120. [CrossRef]
25. De Biasio, A.; Chiariotti, F.; Polese, M.; Zanella, A.; Zorzi, M. A QUIC Implementation for ns-3. In Proceedings of the 2019 Workshop on NS-3, University of Florence, Florence, Italy, 19 June 2019.
26. Xu, L.; Harfoush, K.; Rhee, I. Binary increase congestion control (BIC) for fast long-distance networks. In Proceedings of the IEEE INFOCOM 2004, Hong Kong, China, 7–11 March 2004; pp. 2514–2524.
27. Rossi, D.; Testa, C.; Valenti, S.; Muscariello, L. LEDBAT: the new BitTorrent congestion control protocol. In Proceedings of the 19th International Conference on Computer Communications and Networks, Zurich, Switzerland, 2–5 August 2010; pp. 1–6.
28. Floyd, S.; Henderson, T.; Gurtov, A. The NewReno Modification to TCP's Fast Recovery Algorithm; RFC: Marina del Rey, CA, USA, 2004; pp. 2070–1721.
29. Brakmo, L.S.; O'Malley, S.W.; Peterson, L.L. TCP Vegas: New techniques for congestion detection and avoidance. *ACM SIGCOMM Comput. Commun. Rev.* **1994**, *24*, 24–35. [CrossRef]