Learning-based Congestion Control Simulator for Mobile Internet Education

Junqin Huang, Linghe Kong*, Jiejian Wu, Yutong Liu, Yuchen Li, and Zhe Wang
Shanghai Jiao Tong University, Shanghai, China
{junqin.huang,linghe.kong,wujiejian,isabelleliu,yuchenli,wang-zhe}@sjtu.edu.cn

ABSTRACT

Mobile Internet enables a huge amount of access requests, leading to severe network congestion. To alleviate congestion in the transmission layer, lots of Congestion Control (CC) algorithms have been proposed recently in the research domain, which are specifically designed for various network environments. However, one of the teaching difficulties in mobile Internet education is to allow students to accurately choose the appropriate CC algorithm under the known or measurable network environment.

In this paper, we propose a learning-based CC simulator for mobile Internet education, which provides intuitive suggestions to students on the CC algorithm selections via its learning ability in practical network environments. Our simulator consists of three key modules: the network data module, learning module, and CC module. It has built-in several default CC algorithms and supports students' customized algorithms. The performance of the proposed simulator is evaluated on the implemented simulator prototype with both real and simulated network links. Evaluation results show that the simulator can dynamically select proper CC algorithms in the light of network environments to achieve higher throughput, which benefits students in understanding the working mechanisms of CC algorithms intuitively.

CCS CONCEPTS

• **Computing methodologies** → **Modeling and simulation**; *Artificial intelligence*; *Machine learning*; • **Networks** → **Network algorithms**; • **Applied computing** → *Education*.

*Corresponding author.

KEYWORDS

Congestion control; Learning intelligence; Mobile Internet education; Network simulator

ACM Reference Format:
Junqin Huang, Linghe Kong*, Jiejian Wu, Yutong Liu, Yuchen Li, and Zhe Wang. 2022. Learning-based Congestion Control Simulator for Mobile Internet Education. In *16th ACM Workshop on Mobility in the Evolving Internet Architecture (MobiArch '21), January 31–February 4, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3477091.3482760

1 INTRODUCTION

With the development and popularity of mobile Internet, people can access online resources, e.g., video, game, meeting, and social network, via their mobile devices every time and everywhere, which brings heavy traffic burden to the network. And due to the highly dynamic network environment of mobile Internet, severe network congestion is inevitable, causing poor user experience. In order to alleviate network congestion, recent advances have been made on Congestion Control (CC) algorithms [2, 4]. However, traditional analytical approaches cannot well-explain the algorithm behavior under the complex network environment.

In mobile Internet education or research, network simulators are commonly used to evaluate the performance of CC algorithms in different network environments. There are several popular network simulators, such as ns-3 [11], OMNeT++ [13], OPNET [3]. ns-3 is primarily targeted for research and educational usages and achieves stable performance, however, is hard to use due to the steep learning curve. OPNET provides a flexible GUI to simplify simulation designs in complex scenarios but expensive. Although OMNeT++ offers component-based discrete-event simulator tools, its mobility extension is incomplete. Besides, existing network simulators cannot support customized CC schemes. Specifically, a TCP connection only adopts one CC algorithm regardless of network environment changes. Facing various CC algorithms mentioned above, one of the teaching difficulties in mobile Internet education is to allow students to accurately choose the appropriate CC algorithm under the known or measurable network environment. Therefore, a new and powerful CC simulator should be designed to assist mobile Internet education.

With the explosion of Machine Learning (ML) and Artificial Intelligence (AI) fields, ML and AI[1] technologies powered decision-making systems have been widely used in many fields, including manufacturing, financial modeling, education, marketing, etc [5]. Thus, we consider that it is also promising to leverage learning-based methods to make accurate decisions on switching CC algorithms for TCP connections in different network environments.

In this paper, we propose a learning-based CC simulator for mobile Internet education, which consists of three modules, *i.e.*, network data module, learning module, and CC module. We provide both real and simulated network links in the network data module, and model data links according to three indicators extracted from TCP connections. We divide the learning module into offline training and online decision-making sub-modules, to make accurate decisions on indicators of link models. The CC module embeds several optional CC algorithms and switches the optimal CC algorithm in the light of the decision made by the learning module periodically. Our contributions are summarized as follows:

- We import learning methods into the network simulator, which supports dynamic switching of CC algorithms according to the learning on real network conditions. It assists students to intuitively understand appropriate CC algorithms in different network environments.
- We propose the modular design of the network simulator, including three flexible core modules. It allows students to customize their learning methods and CC algorithms as needed.
- We implemented a simulator prototype and conducted extensive experiments to verify the efficiency of the simulator. Evaluation results show that the CC simulator performs well in highly dynamic mobile network environments, which is practical for research and education.

The rest of this paper is organized as follows. In Section 2, we briefly review related work in network simulators. Section 3 presents preliminaries of CC algorithms and learning methods. We introduce the simulator design in Section 4. In Section 5, we describe the implementation and evaluation of the proposed CC simulator. Finally, we conclude this paper in Section 6.

2 RELATED WORK

We briefly review several popular network simulators or simulation libraries in this section.

ns-3 [11] is a discrete event-based network simulator for Internet systems, primarily targeted for research and educational usage. It supports substantial simulations of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. OMNeT++ [13] is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. It supports internet protocols, wireless networks, switched LANs, and etc. OPNET [3] can simulate the behavior and performance of any type of network. The brightest feature of OPNET lies in its versatility, which allows users to create and simulate different network topologies. NetSim [12] is an end-to-end, full-stack, packet-level network simulator and emulator. It provides users with a developing environment for protocol modeling and simulating, and allows users to analyze networks with low cost and high flexibility.

[1]ML and AI are collectively referred to as *learning* in the rest of this paper.

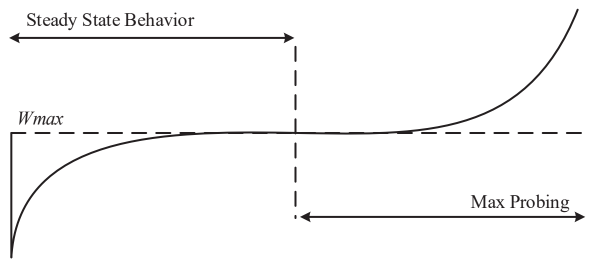

Figure 1: The congestion window growth function of CUBIC.

Compared with the above simulators, we observe that the state-of-the-art simulators cannot integrate *learning* methods for switching of CC algorithms according to network environments.

3 PRELIMINARY

We introduce classical CC algorithms and learning methods in this section for readers to better understand our designs.

3.1 Congestion Control Algorithms

CUBIC and BBR are two classical CC algorithms widely used in Wide Area Networks (WAN) and Data Center Networks (DCN).

3.1.1 CUBIC. CUBIC [4] is the default CC algorithm on current Linux systems. As shown in Fig. 1, its CC window grows per a cubic function, which is designed for better scalability in the current fast and long-haul network environment. The congestion window growth of CUBIC is independent of Round-Trip Time (RTT) to ensure flow-to-flow fairness.

The working mechanism of CUBIC is illustrated as follows. When a packet loss occurs, CUBIC records the congestion window size at this time as W_{max} then performs a multiplicative reduction of the congestion window by a factor β, where β is a window reduction constant. After that, TCP fast recovery and re-transmission are performed as usual. After entering congestion avoidance from the fast recovery phase, the window is increased using the concave contour of the

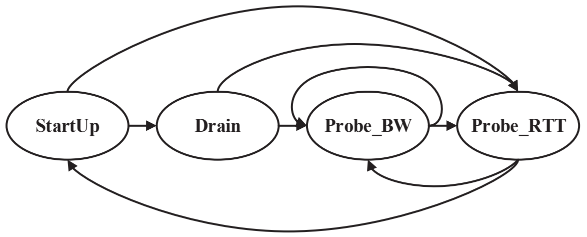

Figure 2: The working state machine of BBR.

cubic function (corresponding to the left part of Fig. 1). The cubic function is set to reach a stabilization point at W_{max} and then the exploration of a new maximal window. If a new maximal window exists, the window is increased using the convex contour of the cubic function.

3.1.2 BBR. Google proposed the BBR [2] CC algorithm in 2016. The core idea of BBR is to use the estimated bandwidth and latency to directly infer the congestion level and then calculate the sliding window. The workflow of BBR algorithm is shown in Fig. 2, which contains four phases: *StartUp*, *Drain*, *Probe_BW*, *Probe_RTT*.

In the *StartUp* phase, BBR will increase the rate of sending packets exponentially to explore the maximal BandWidth (BW) as soon as possible. And then, BBR switches to the *Drain* phase, which drains the queue created during the *StartUp* phase. After that, BBR enters the cruise state and switches between the *Probe_BW* phase and the *Probe_RTT* phase periodically. The *Probe_BW* phase explores a new maximal BW by sending packets according to the rate gain $[1.25, 0.75, 1, 1, 1, 1, 1, 1]$ periodically. The *Probe_RTT* phase drains queue to refresh a new minimal RTT.

3.2 Learning Method

We introduce the learning methods commonly used for decision making in this section.

3.2.1 Decision tree learning. Decision Tree (DT) [7] learning is one of the predictive modeling approaches used in statistics, data mining, and machine learning. It uses a decision tree (as a predictive model) to walk from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Training a DT model can be divided into three steps: feature selection, DT generation, DT pruning. There are three classic DT algorithms, *i.e.*, ID3 [9], C4.5 [10], CART [6]. ID3 is the earliest proposed DT algorithm, which utilizes the information gain to do feature selection. C4.5 makes improvements based on ID3, which uses the information gain ratio as an indicator for feature selection. CART introduces the Gini coefficient to replace the original information entropy model.

3.2.2 Ensemble learning. Ensemble Learning (EL) [14] is commonly used to enhance the generalization of models. The core idea of EL is to train multiple independent base learners, each learner is generated through existing learning methods on given training data. EL uses a specific strategy to aggregate these base learners and makes these learners work together on learning tasks. In EL, even though one base learner makes wrong predictions, other base learners can correct them. Thus, EL-assisted models could achieve a better performance compared to single learning models.

4 SIMULATOR DESIGN

The overview design of the simulator is shown in Fig. 3, which contains three core modules: network data module, learning module, and CC module. The network data module is responsible for simulating network links and extracting the key information from packets steaming to model data links. The learning module receives new modeling features of data links periodically, and makes decisions via trained decision models. And the CC module dynamically adjusts the optimal CC algorithm according to the decisions made by the previous module. The detailed design of these modules is described in the following sections.

4.1 Network Data Module

The network data module consists of two sub-modules, *i.e.*, network link and network environment modeling. The network link sub-module not only offers real network links by using real network traffic datasets as input, but also allows users to construct simulated network links by generating customized network data (*e.g.*, tuning the network bandwidth, data loss rate, RTT). The network environment modeling sub-module obtains packets from the network links and extracts the key information of network streams, such as RTT of TCP connection, the number of ACKs, and then calculates statistics on these key indicators to model the network environment.

In order to model the network environment, we firstly analyze several influence factors that should be concerned with. As a TCP connection may contain heterogeneous network links, such as LAN, WAN, satellites, it is non-trivial to illustrate a TCP connection precisely. Thus, we need to abstract and summarize the network link where the TCP connection is located.

From a macro perspective, even though a TCP connection crosses multiple types of data links, one of the data links may be the bottleneck of the whole TCP connection due to the low throughput, thereby impacting the throughput of data transmission significantly. The bottleneck throughput is expressed by

$$BtlThr = \min(Thr_1, Thr_2, \cdots, Thr_n), \tag{1}$$

where $BtlThr$ denotes the throughput of the bottleneck link, Thr_i denotes the throughput of the i-th data link, which

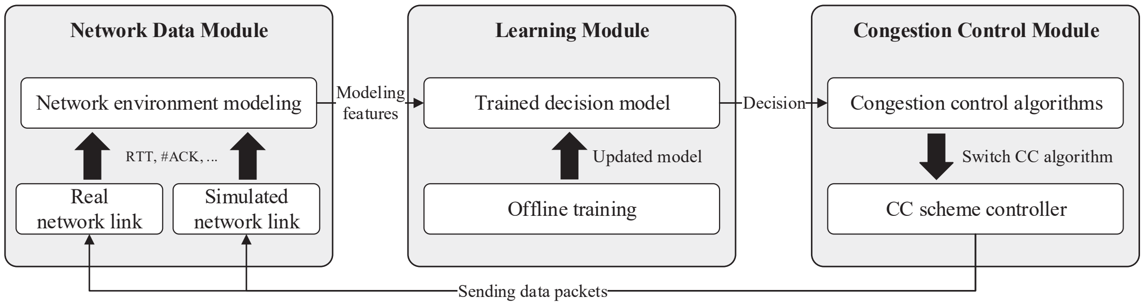

Figure 3: The modular design of the learning-based CC simulator.

can be estimated by the delivery rate of packets. According to Eqn. 1, we observe that the actual throughput of a TCP connection depends on the minimal throughput of all data links, *i.e.*, bottleneck throughput. On the other hand, because the throughput of the bottleneck link is minimal, if the whole TCP connection transmits data in the maximal transmission rate, it may happen that the leaving rate of data packets is smaller than arriving. In this case, those data packets that cannot leave the bottleneck link in time will be stored in the buffer queue of the bottleneck link. When the buffer of the bottleneck link is full, then some data packets will be discarded, which is called a buffer overflow.

Another important factor is the RTT. Different from the bottleneck throughput, RTT of the whole TCP connection is the sum of each data link's RTT, *i.e.*,

$$RTT = \sum_{i=1}^{n} RTT_i, \tag{2}$$

where RTT_i denotes the round-trip time of the i-th data link in the whole TCP connection.

Besides the above two key metrics, the loss rate of links can impact the performance of data transmission. Provided that the buffer is large enough, the system measures the packet loss rate of a link in terms of the number of packets lost as a percentage of the number of packets sent. Packet loss can be caused by the lack of stability of the channel, some fading of the channel, interference during data transmission, defects in processing equipment, etc. In general, the packet loss rate of wired channels is very low, while the random packet loss probability of wireless channels is relatively high. The total packet loss rate $Lossrate$ of the whole data link where the TCP connection is located is affected by the packet loss rate of each segment of the link $Lossrate_i$, which can be calculated as

$$Lossrate = 1 - \sum_{i=1}^{n} (1 - Lossrate_i). \tag{3}$$

Therefore, we consider that we can abstract a complex data link as a bottleneck link δ, and use a triple

$$\delta \leftarrow (BtlThr, RTT, Lossrate) \tag{4}$$

Table 1: Optional data link parameters for generating training datasets.

Parameter	Value
BltThr	0.6, 1.2, 2, 4, 6, 8, 10, 12,16, 20, 24 (Mbps)
RTT	30, 50, 75, 100, 150, 200, 250, 300 (ms)
Lossrate	0, 0.05, 0.1, 0.3, 0.5, 1, 3, 5, 10, 20 (%)

to illustrate the properties of the bottleneck link δ, thereby simplifying the topology of complete data links. Without loss of generality, the triple of modeling features can be extended or customized by other network metrics.

4.2 Learning Module

The learning module consists of two sub-modules, *i.e.*, offline training and online decision-making.

In the offline training phase, the module trains a decision model for decision-making. In the CC simulator, the decision model actually is a classification model. Here we take DT learning as the learning method and CUBIC and BBR as the optional CC algorithms. For constructing the training dataset, we set 8 RTT parameters, 11 $BtlThr$ parameters, and 10 $Lossrate$ parameters, which is shown in Table 1. These parameters are combined to form 880 different triples, *i.e.*, 880 types of network environments. And in each network environment, we evaluate the performance of CUBIC and BBR respectively, totally for 1760 experiments. Each experiment last for 20s, the buffer size is set to 300KB, and we select the CC algorithm which has a larger throughput in the network environment as the label of this network scenario. In order to improve the generalization of the trained model and avoid over-fitting, we also utilize the EL method to aggregate 10 independent DT learning, which achieves a better performance than single DT learning method. Although we take DT learning as an example, users can customize their learning methods flexibly, such as SVM, CNN, Random Forest, etc. In this module, we provide DT learning as the default method.

In the online decision-making phase, the module receives the modeling parameters of the network link, *i.e.*, bottleneck throughput, round-trip time, loss rate, from the network module periodically, and inputs these parameters into the trained decision model for obtaining the decision result.

4.3 Congestion Control Module

The CC module is relatively simple, which contains a set of optional CC algorithms and a CC scheme controller. We provide CUBIC and BBR as default CC algorithms. Users can customize their optional CC algorithms flexibly. Once receiving a new decision from the learning module, the switching command is forwarded to the CC scheme controller, which will apply the new CC scheme to send data packets.

It is noted that the decision interval cannot be too long or too short. If the interval is too long, the effect of switching CC algorithms is not significant and cannot be observed intuitively. If the interval is too short, the CC algorithm may be changed before entering the steady state, which may lead to network chaos and also brings higher computing cost for decision-making. Thus, the decision interval should be finely tuned, which is discussed in Section 5.

5 IMPLEMENTATION AND EVALUATION

5.1 Simulator Implementation

In the network data module, we used Mahimahi [8] to implement the network link, which can adjust $BltThr$, RTT, $Lossrate$ to simulate different network links. We used Orca [1] to realize statistics on the information of sending/receiving packets, such as ACKs, RTT, etc. In the learning module, we used PyTorch as the learning framework, which allows users to customize their learning methods in an easy way. In the congestion control module, we used Orca to implement the CC scheme controller, which can modify the parameter settings of TCP connections.

5.2 Performance Evaluation

We evaluated the performance of the simulator under two types of network conditions, *i.e.*, steady links and dynamic links. In this experiment, we set the running time of the simulator as 20s for steady links, 60s for dynamic links to simulate long TCP streams. The initial CC algorithm is the default one used in Linux, *i.e.*, CUBIC. The buffer size of links is set to 300KB, and the buffer queue adopts the *Droptail* strategy. The decision interval is 4s, *i.e.*, determining whether to change the CC algorithm of a TCP connection every 4s.

Steady links. We evaluated the simulator under the network environment with 50 ms RTT, 10 Mbps $BtlThr$, 0% $Lossrate$, whose result is shown in Fig. 4. We observe that the decision model chose CUBIC in most cases. The reason is that CUBIC can achieve higher throughput than BBR in

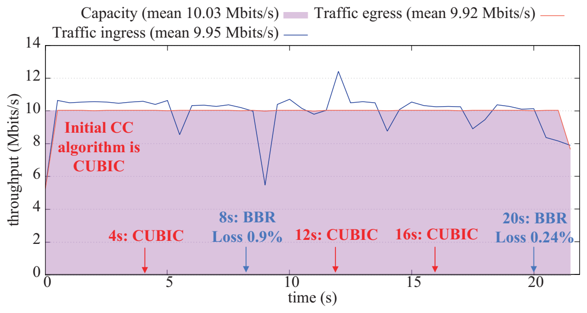

Figure 4: The simulator performance in steady links.

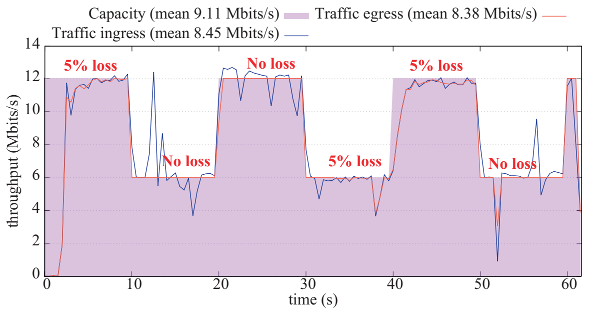

Figure 5: The simulator performance in dynamic links with multiple attributes changing.

a lossless scenario. Thus, the decision-making meets expectations. And we notice that the simulator switched the CC algorithm to BBR in the 8-th second. The reason is that the packet loss rate calculated in the previous decision period is close to 1%, so the decision model considers BBR is more appropriate in current network environment. To this end, the simulator can dynamically adjust the CC algorithm periodically according to the network environment.

Dynamic links. In the dynamic link experiment, we will adjust data link attributes, *i.e.*, the triple, every 10s to simulate the changes of data transmission links. The link attributes changes periodically according to the order

$$\rightarrow (12Mbps, 50ms, 5\%) \rightarrow (6Mbps, 50ms, 0\%)$$
$$\rightarrow (12Mbps, 50ms, 0\%) \rightarrow (6Mbps, 50ms, 5\%) \rightarrow$$

The evaluation result is shown in Fig. 5. We observe that the simulator switched the CC algorithm to CUBIC when the link is lossless, and switched to BBR when the link is lossy, which indicates the decisions made by the simulator are accurate.

To further verify the performance of the simulator, we also evaluated the simulator under real network environments, which is shown in Fig. 6. The real network traffic datasets are sampled from an online audio and video site. We observe that the traffic egress is close to the capacity of links for most of the time, which indicates that the simulator can switch the CC algorithms correctly and timely.

Table 2: Comparison between our proposed simulator and existing network simulators.

Simulator	Intelligence Enabled	Language	GUI Support	Ease of Use	Available Network
Ours	✓	C++ and Python	✗	Easy	Wired, Wireless, Mobile Networks
OMNeT++	✗	C++	✓	Easy	Wired, Wireless, Adhoc and WSN
ns-3	✗	C++ and Python	✓	Hard	Wired, Wireless, Adhoc and WSN
OPNET	✗	C and C++	✓	Easy	Wired, Wireless, Adhoc and WSN
NetSim	✗	C and Java	✓	Easy	Wired&Wireless Sensor Networks

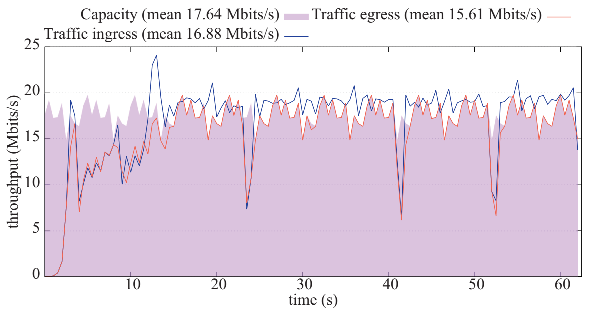

Figure 6: The simulator performance in real network links with bandwidth fluctuation.

Competitor comparison. We compare our simulator with several existing popular network simulators in Table 2. We observe that only our proposed simulator supports learning intelligence for smarter congestion control, which is more suitable for analyzing variable mobile networks. Moreover, our simulator provides flexible API interfaces for students to call default functions or customize their methods, so that it is easy to use. However, our simulator does not have an advantage in terms of market share compared to existing commercial simulators. Since our simulator is still in the initial development stage, it is lack of GUI support, which is one of our future work.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose a learning-based congestion control simulator for mobile Internet education. In particular, we enable learning methodologies in the network simulator for fined decision-making on the selection of CC algorithms, which gives students an intuitive feeling that which CC algorithm performs better under specific network environments. We introduce the systematic design of the simulator, *i.e.*, the three modules, and implemented a prototype and verify the correctness and practicality of the simulator.

In future work, we focus on enhancing the ease of use and features of the simulator. First, we plan to add a visualization interface for the simulator to provide easier operations. Second, we will embed more CC algorithms and learning methods in the simulator as default options, which will be beneficial for fresh students.

ACKNOWLEDGMENTS

This work was supported in part by NSFC grant 61972253, U1908212, 72061127001, the Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institutions of Higher Learning.

REFERENCES

[1] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. 2020. Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet. In *ACM SIGCOMM*. Virtual Event, USA.
[2] 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 (2016), 20âĂŞ53.
[3] Xinjie Chang. 1999. Network Simulations with OPNET. In *ACM WSC*. Phoenix, Arizona, USA, 307âĂŞ314.
[4] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008), 64âĂŞ74.
[5] M. I. Jordan and T. M. Mitchell. 2015. Machine learning: Trends, perspectives, and prospects. *Science* 349, 6245 (2015), 255–260.
[6] Wei-Yin Loh. 2011. Classification and regression trees. *WIREs Data Mining and Knowledge Discovery* 1, 1 (2011), 14–23.
[7] Sreerama K. Murthy. 1998. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery* 2, 4 (1998), 345–389.
[8] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX ATC*. Santa Clara, CA, 417–429.
[9] J. Ross Quinlan. 1979. Discovering rules by induction from large collections of examples. *Expert systems in the micro electronics age* (1979).
[10] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[11] George F. Riley and Thomas R. Henderson. 2010. *The ns-3 Network Simulator*. Springer Berlin Heidelberg, 15–34.
[12] TETCOS. 2021. NetSim-Network Simulator & Emulator. https://www.tetcos.com/ Accessed June 15, 2021. (2021).
[13] Andras Varga. 2010. *OMNeT++*. Springer Berlin Heidelberg, 35–59.
[14] Zhi-Hua Zhou. 2012. *Ensemble methods: foundations and algorithms*. CRC press.