Congestion Control: A Renaissance with Machine Learning

Wenting Wei, Huaxi Gu, and Baochun Li

Abstract

In the past several decades, it has been well known that the Transmission Control Protocol (TCP), even with its modern variants such as CUBIC, may not perform optimally when available bottleneck bandwidth needs to be fully utilized, yet without unnecessarily increasing the end-to-end latency. These observations have led to a recent resurgence of interest in the topic of redesigning congestion control protocols and replacing modern TCP variants using machine learning. In this article, we examine and compare some of the most prominent recent research results on the use of machine learning to redesign congestion control protocols, with an editorial commentary on potential research directions in the near-term future.

Introduction

Congestion control is one of the most fundamental and challenging problems in computer networking research. The challenge involves the design of a protocol that is simple and practical to implement, yet is able to maximize the achieved throughput between a source and a destination, avoiding any potential congestion in the network in between. The first challenge in the design of congestion control protocols lies in the wide variety of network characteristics: the network can be a wireless connection between a smartphone and an airport WiFi access point, shared with hundreds of other smartphones; or it can be a datacenter network operating at 100 Gb/s with very low propagation delays. Both the available bottleneck bandwidth and the propagation delay can vary by a few orders of magnitude, leading to a wide range in terms of the bandwidth-delay product (BDP), defined as the product of the bottleneck bandwidth and the round-trip propagation time, and the congestion control algorithm must be able to operate effectively over the entire range of the BDP.

Figure 1 shows an illustration of the bandwidth-delay product, which represents the "volume" of the "pipe" between the source and the destination at a high level. The ultimate objective when designing a congestion control algorithm is to fill such a "pipe" *just right*: utilizing the bottleneck bandwidth as much as possible, and ensuring that the total data in flight between the source and the destination is equal to the BDP. When both objectives are achieved, queuing delays, i.e. the time spent for packets to wait in queues along the path between the source and the destination, are kept to the minimum. To further exacerbate the problem, the bottleneck bandwidth may vary, as the flows sharing such bandwidth may arrive and terminate at any time. A well-designed congestion control algorithm should allow these flows to share bottleneck bandwidth *fairly*.

In the past several decades, it has been well known that the Transmission Control Protocol (TCP), when used as a congestion control mechanism, has been simple, effective and highly scalable as the scale of the Internet evolves by a few orders of magnitude. Yet, it has also been widely acknowledged that TCP suffers from subpar — some call it notoriously poor — performance when the network links become lossy or when the BDP becomes high.

To address the problems of TCP, the community has resorted to two different directions of research. The first is to redesign congestion control algorithms from scratch, a "clean slate" approach, as a replacement for congestion control in TCP, without prioritizing backward compatibility but still maintaining TCP-friendliness. As a prominent example, the eXplicit Control Protocol (XCP) [1] has been designed specifically for the high-BDP environments as alternatives to TCP.

The second is to "patch" the protocol while maintaining backward compatibility with respect to fairness to the traditional TCP variants such as TCP-Reno. This is exemplified by CUBIC [2], a TCP variant that is designed for high-BDP environments by growing the congestion window more aggressively beyond the saturation point. Since it was proposed, CUBIC has become a remarkable success, making its way into the Linux kernel since version 2.6.19 as the default TCP implementation, as well as the Windows 10 kernel. It has been widely accepted as the *de facto* TCP implementation, and appears to be a preferred choice as compared to other backward-compatible replacements of TCP in the literature, such as XCP and FAST TCP.

From the perspective of the number of papers published each year on the topic of congestion control (Fig. 2), it appears that the popularity of this topic reached a "plateau" around 2007-2008, and trended downward since then. Starting in 2017, however, we observed an uptick of interest in congestion control as a research topic. Such a "renaissance" may not be reflected in the number of papers in the literature yet; but as we will elaborate in this article, it has certainly been reflected in the interest of recent papers to move away from the conventional wisdom of using heuristics to the use of machine learning techniques. They have

Digital Object Identifier:
10.1109/MNET.011.2000603

Wenting Wei and Huaxi Gu are with Xidian University; Baochun Li is with the University of Toronto.

been proposed as alternatives to *replace* TCP, following the first category of a *clean slate* design.

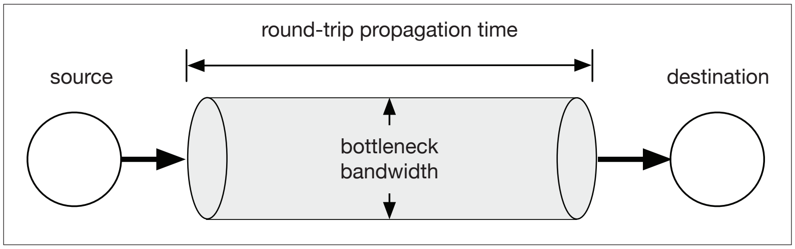
In this article, we present a concise survey of some of the prominent and most influential recent research results on the resurgence of redesigning congestion control protocols by applying machine learning techniques. We will outline several important challenges and their corresponding solutions, and share some of the lessons we have learned. Throughout the article, we will provide insights toward potential future directions as well.

Congestion Control Protocols with Offline and Online Learning

As replacements for TCP variants, it is challenging to design new congestion control protocols that work efficiently in a wide variety of network environments. Again, the ultimate design objectives are to maximize the *utilization* of bottleneck bandwidth, and to ensure that the total data in flight between the source and the destination is equal to the BDP, or equivalently, to reduce queuing delays as much as possible.

BBR: To achieve both objectives, Cardwell *et al.* [3] designed a new congestion-based congestion control protocol, called BBR (Bottleneck Bandwidth and Round-trip propagation time). BBR represents the network path between the source and the destination as a "pipe" with two parameters: the round-trip propagation time (as the "length" of the pipe) and the bottleneck bandwidth (as the minimum diameter). When there is not enough data in flight to fill the pipe, the round-trip propagation time is a fixed value and the delivered rate is low; when the pipe is full, the bottleneck bandwidth dominates the delivery of packets and a queue starts to build at the bottleneck switch. There exists an optimal operating point that maximizes the delivered rate and minimizes queuing delays at the same time. Yet, before BBR was designed, it remains elusive to achieve such an optimal point with a distributed algorithm. In fact, Jaffe *et al.* [4] stated that it was not feasible to design a decentralized protocol to achieve such an optimal point.

BBR, on the other hand, was not as pessimistic, and suggested that such a protocol can be designed by reacting to careful measurements over time. For the round-trip propagation time, BBR proposed to estimate it using the minimum of the round-trip times (RTTs) over a period of time (at the scale of tens of seconds or minutes). For the bottleneck bandwidth, it estimated it using the maximum of the delivery rate over a period of time (typically six to 10 RTTs), which can be accurately measured when acknowledgments are received. Once both parameters are estimated, the BDP can then be computed.



AFIGURE 1. The bandwidth-delay product (BDP) is the product of the bottleneck bandwidth and the round-trip propagation time. An ideal congestion control algorithm will fully utilize the bottleneck bandwidth, yet keeping queuing delays to the minimum by not overfilling the "pipe."

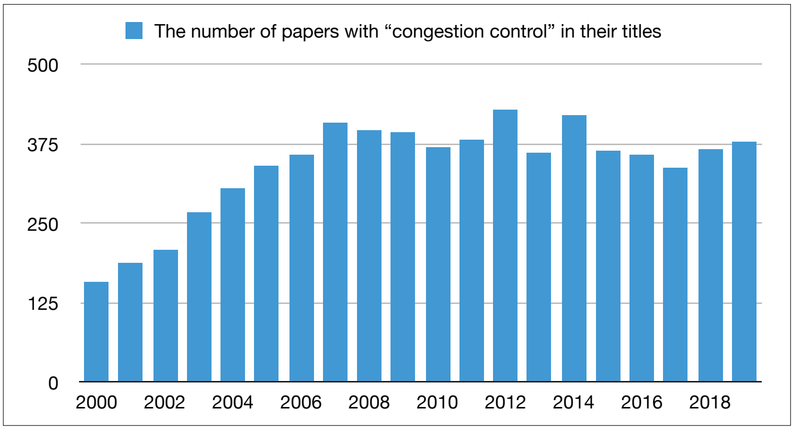


FIGURE 2. The "popularity" of congestion control as a topic, approximated by the number of papers each year with the term *congestion control* in their titles (data obtained from Google Scholar).

When a flow starts, BBR enters a startup state and uses a binary search for the bottleneck bandwidth, roughly doubling the sending rate (with a gain of 2/ln2) when the delivery rate measured keeps increasing. Once the bottleneck bandwidth is reached, it will enter a drain state to drain the queue. At steady state, BBR pays little attention to packet losses, as opposed to TCP variants that it replaces, which react to such packet losses as congestion signals. Instead of continuing to ramp up until packet losses occur, BBR paces packets to be sent so that the data in flight is bounded by a small multiple of the computed BDP. Periodically, it enters a probeBW state to probe for a higher bottleneck bandwidth; and when the round-trip propagation time starts to increase, it enters the drain state again to properly drain the queue.

BBR, as well as its recent upgrade called BBRv2 (BBRv2: Alpha/Preview Release — https://github.com/google/bbr/blob/v2alpha/README.md), has been extensively tested in production systems (e.g., Dropbox Evaluating BBRv2 on the Dropbox Edge Network — https://dropbox.tech/infrastructure/evaluating-bbrv2-on-the-dropbox-edge-network). It has also been made available as a part of the Linux kernel (since version 4.9), FreeBSD, and Chrome. BBR flows are able to converge to their fair shares when competing with other BBR flows, but may not be fair to TCP variants, such as CUBIC (as long as the buffer sizes at intermediate switches are not too large). While both BBR and CUBIC saturate the bottleneck bandwidth, BBR flows experience significantly shorter queuing delays, thanks to the drain state that BBR flows need to cycle through.

Though BBR uses a hand-tuned heuristic rather than machine learning, it nevertheless offers an excellent benchmark that all congestion control protocols designed with machine learning should be compared with.

Copa: Copa [5] represents another noteworthy heuristic that applies a similar design philosophy as BBR. Rather than adjusting the sending rate, Copa adjusts the size of the congestion window (cwnd) instead, as it optimizes an objective function, $\log\lambda - \delta\log d$, that combines a flow's average throughput λ and the packet queuing delay *d*, where δ is a tunable parameter. The goal is to maximize the objective function by seeking to achieve a steady-state sending rate, called the *target rate*, to be inversely proportional to the measured queuing delay.

Since both Copa and BBR use queuing delays rather than packet losses as the congestion signal, they both strive to minimize the packet queuing delays. To minimize queuing delays, Copa and BBR drain the packet queue periodically. Yet, by using a default choice of δ = 0.5 intended for low-latency applications, Copa trades off a small amount of throughput to achieve significantly lower delays compared to BBR. Copa is slightly less aggressive than BBR, and uses two different modes with explicit switching in order to achieve fairness to competing TCP flows sharing the same bottleneck bandwidth. Over certain network environments such as satellite links, Copa is able to outperform BBR significantly in both achievable throughput and average queuing delay.

Remy: For a new congestion control protocol to cope with link capacities that span (approximately) 12 orders of magnitude in the Internet, it is conceivable that optimization and machine learning techniques can be used to *learn* and *optimize for* the dynamic behavior of the network path between a source and a destination, rather than using a hand-tuned heuristic such as BBR.

As one of the first attempts toward this direction, Winstein *et al.* [6] designed a new mechanism, called *Remy*, to generate new congestion control protocols using a data-driven approach. Based on a pre-specified objective for congestion control, a set of assumptions for the desired protocol and models for both the network and the traffic, *Remy* generates and deploys an algorithm that is optimized offline beforehand as a part of an existing TCP source.

A Remy-generated algorithm involves three states as its *memory* (initialized to all zeros at the start of a flow): a moving average of the *interarrival time* between new acknowledgments received; a moving average of the time between TCP source timestamps reflected in those acknowledgments; and the ratio between the most recent RTT and the minimum RTT. The objective function of such an algorithm will avoid building up queues, reducing both packet losses and queuing delays as much as possible.

As each acknowledgment is received, a Remy-generated algorithm updates its memory and then performs a table lookup to find the corresponding action. An action can be an increase (or decrease) of the congestion window (either multiplicatively or additively), or pacing the successive outgoing packets by setting a lower bound on the time between them.

In its offline optimization phase, Remy pre-computes the lookup table by finding the mapping that maximizes the expected value of the objective function, using simulations of various network models with parameters drawn within the ranges of the supplied assumptions. These parameters include the link rates, delays, and the number of sources. It was shown that Remy-generated algorithms outperformed TCP variants such as CUBIC, and protocols that require modifications to intermediate switches, such as XCP [1].

PCC: To avoid hardwired mapping between congestion control actions and packet loss events in traditional TCP variants, Dong *et al.* [7] proposed Performance-oriented Congestion Control (PCC), a new congestion control architecture that learns based on live experimental evidence. Different from Remy, PCC learns in an online fashion using multiple *micro-experiments*, each sending at two different rates, and subsequently moving in a direction that leads to better performance. Its key idea is to learn the relationship between rate control actions and the performance that it empirically observed. The performance is characterized by a utility function that describes an objective, such as to achieve a high utilization of the bottleneck bandwidth with low loss rates.

In each micro-experiment, a PCC source sends at two different rates: one is marginally higher than the current rate, and the other is marginally lower. If it finds that one of these directions leads to better performance (characterized by a utility function), it will be selected as the sending rate for the next micro-experiment, and continues in this direction as long as the utility continues increasing. Equivalently, over multiple micro-experiments, we can claim that PCC runs an online learning algorithm in the spirit of gradient ascent. Online learning eliminates a major disadvantage of Remy's offline optimization: when the actual network environment deviates from the input assumptions and network models made, performance may degrade due to such a mismatch.

PCC Vivace: Continuing their own work on performance-oriented congestion control, Dong *et al*. [8] proposed a *Vivace* variant of PCC. Similar to *PCC*, the source in PCC Vivace tests a pair of rates and computes the corresponding values of utility respectively. A significant difference in PCC Vivace, however, is its utility function, which incorporates not only throughput and loss rates as in PCC, but also RTT gradients. Unfortunately, decisions on how the sending rate should be changed in its online learning based on gradient ascent is still challenging, due to the trade-off between the speed of convergence to a steady state and the stability of sending rates. To address this challenge, PCC Vivace employs an elaborate heuristic that converts utility gradients (computed in its micro-experiments with a pair of sending rates) to a change in sending rates, with a conservative initial conversion factor to start with, and increasing the conversion factor as it gains more confidence.

With these improvements in protocol design, it can be proved that a stable global rate allocation exists as a Nash equilibrium, especially when multiple flows from sources with different utilities compete for bottleneck bandwidth. As a result, PCC Vivace is more TCP-friendly, converges faster, and reacts more swiftly to changes in network conditions.

Pantheon and Indigo: More recently, Yan *et al*. [9] presented *Pantheon*, an experimental testbed for evaluating and comparing new congestion control protocols fairly. Pantheon is an open-source testbed with a growing collection of congestion control protocols built-in, each verified to compile and run using a continuous-integration system.

One of the evaluated protocols in Pantheon, a hidden treasure, is called *Indigo*, which once again employs a congestion control protocol based on machine learning (ML). As shown in PCC and PCC Vivace, it is difficult to use online learning to train a congestion control algorithm, since ML algorithms require excessively long training times (ranging from hours to weeks), while the condition of network paths evolves in much shorter time scales (seconds). As a result, Indigo employs offline training using Pantheon's abundant supply of network emulators.

Instead of changing the sending rates in existing congestion control protocols (e.g., BBR, Remy, PCC, or PCC Vivace), Indigo observes the network states and adjusts the *congestion window*, which represents the amount of data in flight between the source and the destination. The network state, similar to Remy's *memory*, is characterized by:

- A moving average of the queuing delay (which, in a similar vein as BBR, is measured as the difference between the current and the minimum RTT)
- A moving average of the sending rate
- A moving average of the delivered rate (again similar to BBR)
- The current congestion window size.

Rather than using a heuristic for adjusting sending rates with gradient ascent in PCC Vivace, Indigo uses a Long Short-Term Memory (LSTM) recurrent neural network (RNN) to store the mapping from states to actions, and trains such a RNN using an offline training phase. Once trained and deployed, the mapping will be fixed.

However, as both Remy and Indigo employ offline training, they suffered from the same disadvantage that PCC and PCC Vivace avoided: the training environment should not deviate from the actual network significantly. If such an assumption is valid, it has been shown in extensive experiments that Indigo outperforms existing online algorithms, such as BBR, PCC, and PCC Vivace, in terms of achieved throughput and end-to-end delays. This comes as no surprise: if the training phase uses an emulator that characterizes the actual network precisely, the offline-trained RNN will bypass the time-consuming online learning process in BBR, PCC, and PCC Vivace, making more accurate decisions over time. Though Indigo's speed of convergence to the steady state and fairness to existing congestion control protocols, such as CUBIC, have not been evaluated, it remains enlightening to see how offline training and recurrent neural networks can significantly improve the performance of congestion control.

Table 1 summarizes some of the common features and key differences across BBR, Copa, Remy, PCC, PCC Vivace, and Indigo. In addition, Fig. 3 provides an illustrative comparison between BBR, Copa, Remy, PCC Vivace, and Indigo, with respect to the performance metrics they take into account, as well as the actions taken by these protocols. From a practical perspective, Google's BBR is a simple yet effective heuristic, and since it has already been incorporated into recent Linux kernel releases since early 2017, it is poised to become a prominent replacement of TCP variants in practice. From a research perspective, we have observed a consistent trend toward using learning techniques to train specific models for different network conditions, and such learning can be performed in an offline or online fashion.

Algorithm	Tuning knobs	Machine learning	Design principles	Algorithm design
BBR [3]	Rate-based	None	Minimizing queuing delays	Hand-tuned heuristic
Copa [5]	Window-based	None	Minimizing queuing delays	Hand-tuned heuristic
Remy [6]	Rate-based	Offline	Based on network models	Lookup table
PCC [7]	Rate-based	Online	No network models needed	Gradient ascent
PCC Vivace [8]	Rate-based	Online	No network models needed	Gradient ascent
Indigo [9]	Window-based	Offline	Based on emulated network models	LSTM RNN

TABLE 1. Recent congestion control protocols designed as replacements to TCP variants.

Congestion Control Protocols with Deep Reinforcement Learning

While new congestion control protocols using machine learning techniques have shown promise adapting to dynamic environments, both offline and online learning algorithms exhibited their inherent disadvantages. Offline learning requires a dedicated offline training phase, and may not perform well if the actual network differs remarkably from the emulated one where offline training was carried out.

Online learning, such as PCC and PCC Vivace, and to some extent even traditional heuristics such as BBR and Copa, requires the design of elaborate hand-tuned heuristics to balance the trade-off between rapid convergence to a high BDP and potentially higher queuing delays and loss rates. For example, Copa adapts its rate exponentially so that it can scale up to high-BDP networks. This is in addition to meeting standard requirements of congestion control, such as utilizing the bottleneck bandwidth fully, and sharing it fairly with other competing flows.

With the recent popularity of deep reinforcement learning (DRL), it is conceivable that it may provide a fully-automated mechanism to train a DRL agent by interacting with a real-world network environment, and to avoid hand-tuned heuristics as much as possible. A DRL agent uses both the *states* observed from the environment and a scalar *reward* to train its deep neural network model, which is the agent's strategy that it uses to produce an action, called its *policy*. In the context of congestion control, the action to be taken by the DRL agent may be an increase in the sending rate or the size of the congestion window. The objective of the DRL agent is to train a policy that maximizes the expected cumulative reward. If DRL is used for congestion control, the hypothesis would be that a well-designed reward function and a curated set of states can be used to train the DRL agent effectively by learning from the actual network environment, more so than hand-tuned heuristics.

In the recent research literature, it was proposed that DRL algorithms can be used to perform *offline* training without relying heavily on the design of heuristics. This is, as we mentioned, due to the fact that a DRL agent is able to continuously evaluate value functions of the environment, and adjusts its actions taken based on well-designed rewards in a feedback loop.

Aurora: Jay *et al*. proposed Aurora [10], a rate-based congestion control protocol based on DRL. The agent in Aurora uses changes in the sending rate as its actions, and uses statistics about latencies, as well as the ratio of packets sent to those acknowledged, as its states. The reward function Aurora uses is formulated as a simple linear function: 10 · throughput — 1000 · latency — 2000 · loss. The DRL agent is trained using Proximal Policy Optimization (PPO).

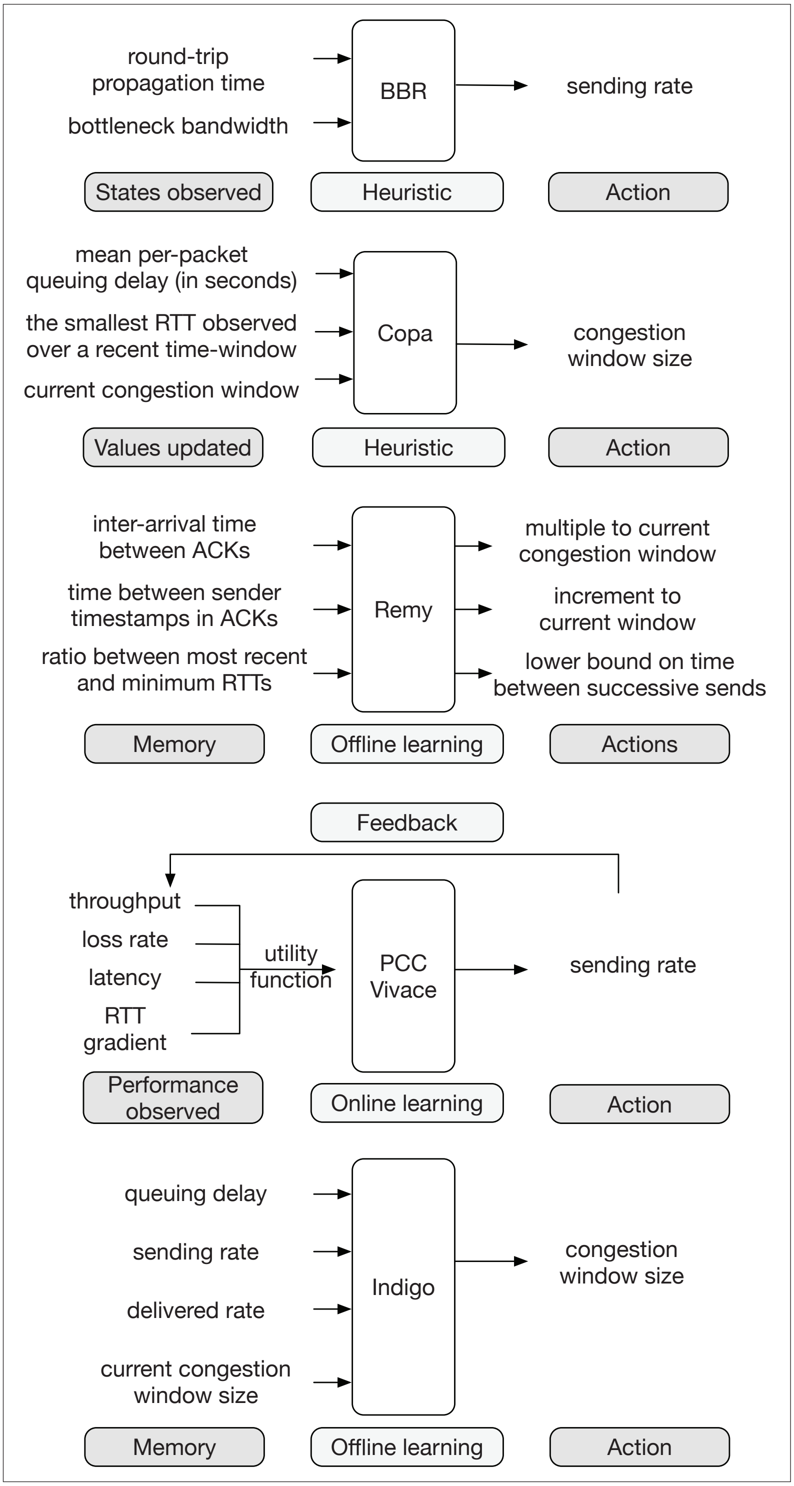


FIGURE 3. Performance metrics used and actions taken in (1) BBR [3]; (2) Copa [5]; (3) Remy [6]; (4) PCC Vivace [8]; and (5) Indigo [9].

Aurora has been evaluated experimentally over Pantheon, with its DRL agent trained in an offline training phase, on links with bandwidth between 1.2 and 6 Mb/s, and latencies varying between 50 and 500 milliseconds. Over a network link whose capacity varies every five seconds to a randomly chosen value in the range of 16 to 32 Mb/s, it has been shown that Aurora outperforms one of the most popular TCP variants, CUBIC, and on par with the best hand-tuned heuristics (BBR and Copa) and PCC Vivace, arguably the best protocol using online learning.

However, Aurora suffers from a number of drawbacks. There are no results presented on any other types of network links (such as LTE or high-BDP networks), and it is debatable whether an artificially simulated link with arbitrary yet periodic capacity variations would be a good representative of real-world networking environments. Furthermore, how Aurora reacts to competing flows, running either TCP CUBIC or Aurora, has not been experimentally evaluated, which is important to show how it performs with respect to fairly sharing the bottleneck bandwidth.

R3Net: R3Net [11] was also proposed by Microsoft to use DRL to design a congestion control protocol, again with a focus on minimizing packet latencies as its design targets video streaming and real-time conferencing applications. It uses a simulator based on trace replays to train the DRL agent using simulated network links and cross traffic. Different from Pantheon, its simulator is able to run 1000x faster than real-time, which helps speed up the training process.

The state in R3Net is a vector containing the delivered rate, the average packet interval, loss rate as a percentage, and the average RTT. The neural network model it uses consists of both fully-connected and Gated Recurrent Unit (GRU) layers, and the action R3Net generates is a sending rate in the range of 0 to 8 Mb/s. The reward function takes into account the delivered rate, RTT, and the loss rate, and is defined as $0.6 \ln 4\mathcal{R} + 1 - \mathcal{D} - 10\mathcal{L}$, where $\mathcal{R}$ is the delivery rate in Mb/s in a time step of 50 milliseconds, $\mathcal{D}$ is the average RTT in seconds, and $\mathcal{L}$ is the packet loss rate. The training algorithm R3Net uses is again PPO, with a learning rate of 3×10^{-5}. Its designs of the reward function and the training algorithm are both very similar to Aurora.

However, different from Aurora, R3Net was designed specifically for low-latency real-time traffic, and was not tuned for the general Internet. It was not evaluated against existing heuristics such as BBR or Copa, or state-of-the-art learning-based protocols such as PCC Vivace. As such, its performance in the general case is not clear.

MVFST-RL: Another DRL-based congestion control protocol [12], proposed by Facebook AI Research, is worthy of some discussions as well. It proposes to use a non-blocking DRL agent, where a sender does not need to wait for the agent, even for a few millisecond, to produce an action. It has been experimentally shown that RL agents

Algorithm	Tuning knob	Reward	Neural network model	Training algorithm
Aurora [10]	Rate-based	Linear function of throughput, latency, and loss	Fully-connected	PPO
R3Net [11]	Rate-based	Function of the receive rate, the average RTT, and loss rate	RNN with GRUs	PPO
MVFSF-RL [12]	Window-based	Linear function of the average throughputand the maximum delay	LSTM and fully-connected	IMPALA with V-trace (asynchronously)
Eagle [13]	Rate-based	Function of goodness, latency, and loss	LSTM	Cross-entropy method
DRL-CC [14]	Window-based	Unspecified utility function of MPTCP flows	LSTM	DDPG
Orca [15]	Window-based	Function of throughput, latency, and loss	RNN	DDPG variant

TABLE 2.Congestion control protocols designed with deep reinforcement learning.

that block the sender would incur a penalty with respect to the number of bytes transmitted over the same duration of time.

In MVFST-RL, the state space includes typical performance metrics such as RTT, queuing delay, packets sent, acknowledged, and lost, as well as a history of recent actions. The actions include additive and multiplicative updates to the size of the congestion window, and for this reason MVFST-RL is essentially a window-based congestion control protocol. The reward is a function of measured throughput and delay: $t - \beta d$, where t is the average throughput in MB/sec, and d is the maximum delay in milliseconds during a window of 100 milliseconds.

The neural network model that MVFST-RL uses is a standard two-layer full-connected network with ReLU no-linearity. The extracted features and the reward are first fed into a single-layer LSTM network, without which the performance would be substantially worse. MVFST-RL integrates Facebook's implementation of the QUIC transport protocol, and decouples the network thread from the RL agent thread, so that gradient updates can be performed in a non-blocking fashion.

Though MVFST-RL shows performance levels competitive with traditional heuristics such as CUBIC and Copa, it is not clear how it compares with alternatives based on offline or online learning, such as Remy or PCC Vivace. It was reported that MVFST-RL faces challenges generalizing over networks with widely different ranges of throughput and delay, as it struggles to achieve high throughput after being trained with low-capacity networks.

Orca: Orca was recently proposed as a *hybrid* congestion control protocol that depends on TCP for fine-grained control actions, and engages a DRL agent to adjust the size of the TCP congestion control window with a coarser time granularity.

Just like existing congestion control protocols that use DRL, Orca also needs to design its state space, its action space, its reward function, as well as its neural network model and training algorithm. Its state space includes average values of delivered rates, packet loss rates, and average delays, among other metrics. Its action space reflects a multiplicative factor 2^{α}, where $-2 < \alpha < 2$, to be applied to the current congestion window size in TCP.

Orca's reward function is defined as a ratio of the current *power* over the maximum one, where *power* is defined as the ratio of the delivered rate over the delay. Packet losses are used as a discount factor when calculating the delivered rate.

Finally, Orca chooses a recurrent network as its neural network model, because the agent does not have direct knowledge of the exact network statistics at the current time, and partial information from the past must be considered. The training algorithm Orca uses is an actor-critic method, operating in a continuous action space using a twin delayed Deep Deterministic Policy Gradient (DDPG) algorithm, which is designed to work well with continuous action spaces.

Performance-wise, after only six hours of training using emulated network environments, Orca is able to achieve the best performance to date in a variety of typical network environments, compared with both traditional hand-tuned heuristics (such as BBRv2) and recent DRL-based congestion control protocols (such as Aurora). It also incurs very little computation overhead, on par with hand-tuned heuristics such as TCP CUBIC and BBR. With respect to fairness, it is friendly to competing TCP CUBIC flows, most likely because it uses TCP CUBIC itself as the underlying congestion control protocol, and therefore does not show aggressive behavior when trying to saturate the available bottleneck bandwidth.

While there exist several alternative congestion control protocols based on DRL in the literature (such as Eagle [13] and DRL-CC [14]), the protocols we have sampled in this article are able to represent the state-of-the-art fairly well. For a summary comparison between existing DRL-based congestion control protocols, refer to Table 2 for key differences between Aurora [10], R3Net [11], MVFSF-RL [12], Eagle [13], DRL-CC [14], and Orca [15]. In addition, Fig. 4 provides an illustrative comparison with respect to the performance metrics that they included in their state spaces, as well as the actions they are designed to take.

Challenges and Future Directions

It has been both surprising and inspiring to observe that work on congestion control, one of the most fundamental problems in computer networking research, has been continuing actively, more than four decades after Cerf and Kahn published their seminal paper on TCP in 1974. Recent works on this topic have seemingly converged on the use of either offline or online learning algorithms to replace heuristics designed for a wide variety of network environments, from high-BDP links across continents to LTE networks. The specific emphasis in the recent literature is on the use of deep reinforcement learning techniques in designing congestion control protocols, in order to learn from the time-varying characteristics of network environments automatically.

Open Challenges: Although the new generation of congestion control protocols based on DRL are promising and quite exciting, there still exist several open questions that we do not yet know definitive answers for. DRL agents are intended to be trained, and as such they are more agile adapting to unseen and more challenging network environments compared to hand-tuned heuristics. Yet, it is not well and fully understood how DRL-based congestion control protocols can be explicitly designed to achieve such a goal.

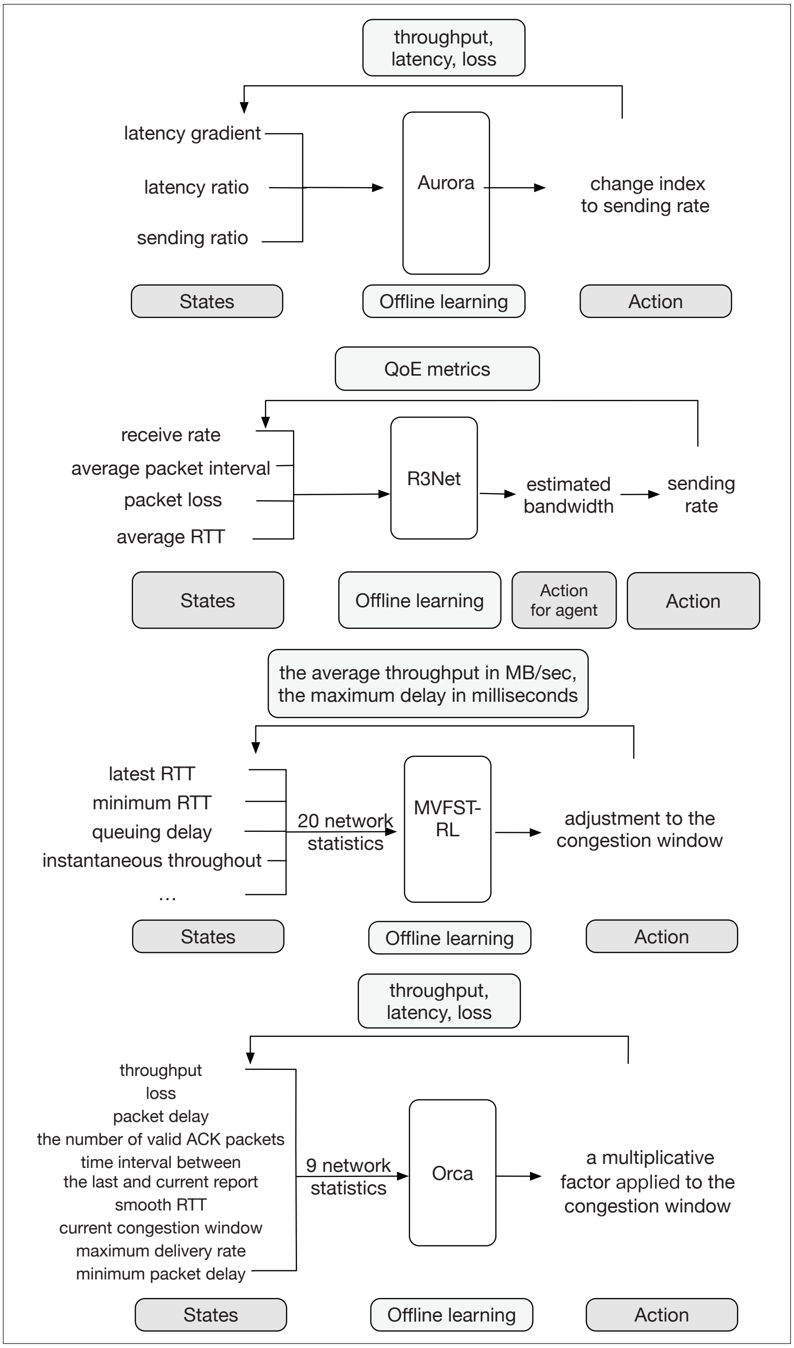


FIGURE 4. Performance metrics used and actions taken in (1) Aurora [10]; (2) R3Net [11]; (3) MVFSF-RL [12]; and (4) Orca [15].

While hand-tuned heuristics, such as BBR, can follow a disciplined approach in their designs following a particular design philosophy, it may not be feasible to do so when tuning DRL agents. We may have to resort to simple trial-and-error explorations, which are both time-consuming and unpredictable in the quality of our design. To further exacerbate the problem, there are many components in a DRL agent that need to be well-designed, including its training algorithm, its neural network model, its state and action spaces, as well as its reward function. To some extent, even more components need to be hand-tuned compared to traditional heuristics, without intuitive links between the designs and their corresponding outcomes.

Finally, one of the most visible roadblocks to realistic deployments of DRL-based congestion control is the feasibility of implementing such protocols efficiently. Congestion control protocols reside in the transport layer, and are traditionally part of the operating system kernel. As examples, TCP CUBIC and BBR are parts of the Linux kernel. There are many practical limitations on what can be implemented in the kernel; TCP CUBIC, for example, uses approximation algorithms to avoid floating-point computation in the kernel due to these limitations. DRL algorithms, however, require computational power that may have to be carried out in user space. Orca, for example, implements its DRL agent in user space with TensorFlow, and communicates with TCP CUBIC in the kernel via socket options. These implementation challenges, together with the performance overhead they may impose, may need to be solved satisfactorily before we see widespread adoption of DRL-based congestion control protocols.

Future Directions: As one of the future research directions, the fact remains that training neural networks to make correct decisions may take a much longer time scale to complete (at least hours), while data flows complete at a much shorter time scale (typically seconds). How do we mix the best "ingredients" of both online and offline learning? Some of the recent machine learning techniques, transfer learning and meta-learning, may become quite promising to bridge such a gap. With these new techniques, a pre-trained model is reused as a starting point for each new network environment, allowing rapid progress adapting to new and unfamiliar network conditions. With some luck, this may just be the key to solve our dilemma of applying learning techniques in designing congestion control protocols that adapt well to a wide variety of network conditions.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under Grant 2018YFE0202800; the National Natural Science Foundation of China under Grant 61634004 and 61934002; the Natural Science Foundation of Shaanxi Province for Distinguished Young Scholars under Grant 2020JC-26; the Fundamental Research Funds for the Central Universities under Grant XJS200119 and JB190105; the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing under Grant 2019A01; and the China Postdoctoral Science Foundation 2018M633465. This work was supported by The Youth Innovation Team of Shaanxi Universities.

References

[1] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," *Proc. ACM SIGCOMM*, 2002.

[2] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating Systems Review*, 2008.

[3] N. Cardwell *et al.*, "BBR: Congestion-Based Congestion Control," *Commun. ACM*, vol. 60, no. 2, Feb. 2017, pp. 58–66.

[5] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," *Proc. 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Renton, WA, 2018, pp. 329–42.

[6] K. Winstein and H. Balakrishnan, "TCP ex Machina: Computer-Generated Congestion Control," *Proc. ACM SIGCOMM*, 2013, pp. 123–34.

[7] M. Dong *et al.*, "PCC: Re-Architecting Congestion Control for Consistent High Performance," *Proc. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015, pp. 395–408.

[8] M. Dong *et al.*, "PCC Vivace: Online-Learning Congestion Control," *Proc. 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018, pp. 343–56.

[9] F. Y. Yan *et al.*, "Pantheon: The Training Ground for Internet Congestion-control Research," *Proc. USENIX Annual Technical Conference (ATC)*, 2018, pp. 731–43.

[4] J. Jaffe, "Flow Control Power Is Nondecentralizable," *IEEE Trans. Commun.*, vol. 29, no. 9, 1981, pp. 1301–06.

[10] N. Jaya *et al.*, "A Deep Reinforcement Learning Perspective on Internet Congestion Control," *Proc. 36th Int'l Conf. Machine Learning (ICML)*, 2019.

[11] J. Fang *et al.*, "Reinforcement Learning for Bandwidth Estimation and Congestion Control in Real-Time Communications," *Proc. NeurIPS Workshop on Machine Learning for Systems*, 2019.

[12] V. Sivakumar *et al.*, "MVFST-RL: An Asynchronous RL Framework for Congestion Control with Delayed Actions," *Proc. NeurIPS Workshop on Machine Learning for Systems*, 2019.

[13] S. Emara, B. Li, and Y. Chen, "Eagle: Refining Congestion Control by Learning from the Experts," *Proc. 39th IEEE Conf. Computer Commun. (INFOCOM)*, 2020, pp. 676–85.

[14] Z. Xu *et al.*, "Experience-Driven Congestion Control: When Multi-Path TCP Meets Deep Reinforcement Learning," *IEEE JSAC*, vol. 37, no. 6, 2019, pp. 1325–36.

[15] S. Abbasloo *et al.*, "Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet," *Proc. ACM SIGCOMM*, 2020, pp. 632–47.

Biographies

Wenting Wei received the M.E. and Ph.D. degrees in telecommunication and information systems from Xidian University in 2014 and 2019, respectively. Since 2019, she has been working at the State Key Lab of ISN, Xidian University. Her main research interests include data center networking, network virtualization, cloud computing and intelligent transportation.

Huaxi Gu received the B.E. and Ph.D. degrees in telecommunication engineering and telecommunication and information systems from Xidian University in 2000 and 2005, respectively. He is a full Professor at the State Key Lab of ISN, Telecommunication Department at Xidian University. His current interests include networking technologies, network on chip and optical interconnect.

Baochun Li received his B.Engr. degree from Tsinghua University in 1995, and his M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign in 1997 and 2000, respectively. He is a professor in the Department of Electrical and Computer Engineering at the University of Toronto. His research interests include large-scale distributed systems, cloud computing and wireless networks.