C3: Learning Congestion Controllers with Formal Certificates

Submission #590, 12 pages body, 16 pages total

Abstract

Learning-based congestion controllers offer better adaptability compared to traditional heuristic algorithms. However, the inherent unreliability of learning techniques can cause learning-based controllers to behave poorly, creating a need for formal guarantees. While methods for formally verifying learned congestion controllers exist, these methods offer binary feedback that cannot optimize the controller toward better behavior. We improve this state-of-the-art via C3, a new learning framework for congestion control that integrates the concept of "formal certification in the learning loop". C3 uses an abstract interpreter that can produce robustness and performance certificates to guide the training process, rewarding models that are robust and performant even on worst-case inputs. Our evaluation demonstrates that unlike state-of-the-art learned controllers, C3-trained controllers provide both adaptability and worst-case reliability across a range of network conditions.

1 Introduction

Effectively controlling congestion in networks is a longstanding problem in computer systems. Several recent approaches to this problem use neural models, trained using reinforcement learning (RL), as adaptive congestion controllers. In typical network environments, such neural models have been shown to outperform classical hand-written controllers frequently [2, 14, 27, 50].

However, the unreliability of neural models raises concerns about their deployment in real-world performance-critical networks. As we demonstrate in Section 2, established neural congestion controllers can exhibit highly suboptimal behavior on unforeseen inputs; for example, they are *not robust* to noise in network state measurements, and some decisions they make can drag the transport connection toward extremely *poor performance*. At the same time, neural models are opaque and, unlike manually designed controllers, cannot be debugged using conventional software engineering techniques.

Formal verification is a natural way to address these reliability challenges. In particular, there have been several recent efforts to formally verify congestion controllers [3, 5, 16, 20, 44], including learned ones [16]. A key benefit of formal methods is that they provide not only a strong assertion that a controller behaves well on all possible inputs, but a *sound certificate* that proves that it does so.

However, attempts to formally verify learned congestion controllers face an immediate challenge: controllers trained using standard performance objectives can violate key robustness and performance requirements *in the worst case*. In case verification fails, we are left with no option but to train an entirely "new" controller that may fail all over again.

In this paper, we address this challenge through a learned congestion control approach, called C3, that *integrates verification and learning*. Specifically, we present a verification procedure where a verifier does not just provide a boolean decision about whether a controller satisfies a desired property, but *quantitative feedback* on how far the controller is from provably satisfying the property. We also present a learning algorithm that uses this feedback as part of the training loss, ensuring that training iterations drive the learned controller towards *high worst-case satisfaction of properties* in addition to high average-case performance. The output of this learning algorithm is a neural controller and a *certificate*, or proof, that the controller satisfies its desired properties.

We apply the C3 framework to Orca, a state-of-the-art RL-based congestion controller [2]. Our verifier is based on abstract interpretation, a classic approach for propagating symbolically represented sets of inputs through systems. This propagation covers all the possible mappings from the inputs (network states used as features) to the output (congestion window to use) that the learning-based congestion controller can induce. During each training step, we compute the distance between the set of outputs computed via abstract interpretation (which includes the worst-case outputs) and the set of outputs that satisfy a specified property. We then use this distance to shape Orca's RL reward function.

In summary, our key contributions are:

- We motivate the need for formal certificates for learned congestion controllers — and, more generally, for ML-controlled software systems.
- We present C3, the first approach to building ML-controlled systems that integrates learning with "formal certification in the loop."
- We present an implementation of our approach built on top of Orca, a state-of-the-art neural congestion controller. We define and provide certification of two key properties – one relating to good performance and another to robustness.
- We evaluate C3 and Orca across synthetic and real-world network traces, demonstrating the benefits of "formal certificate in-the-loop training" – our framework produces models with sound certificates for performance- and robustness-related properties on large subsets of the input space.

Our experiments show that a C3 model trained with a performance property provides up to 78% smaller p95 queueing delays compared to Orca, while sustaining comparable bandwidth utilization; and consistently provides up to 84% smaller p95 delays compared to TCP Cubic and up to 7% higher average bandwidth utilization compared to TCP Vegas. Similarly, a C3 model trained with a robustness property is significantly more robust to the worst-case noise compared to Orca, while maintaining good average-case performance.

2 Motivation and Overview

We begin by demonstrating scenarios where a state-of-the-art learned congestion controller (LCC) behaves suboptimally. Next, we sketch our approach to addressing these issues.

2.1 Issues with Existing LCCs

Traditional congestion control algorithms like TCP Cubic [22] are human-designed. As a result, their behaviors are well-understood, with clearly defined and rigorously analyzed worst-case bounds [5, 10, 42, 44]. In contrast, the emergence of LCCs [2, 14, 27, 50] presents new challenges. While these algorithms show promise in adapting to complex and dynamic network environments, they often operate as black boxes. The lack of transparency in their decision-making makes performing a similar analysis of their worst-case behavior difficult.

We use Orca [2], an existing LCC as the basis for analysis in this paper[1]. Orca is a deep reinforcement learning (RL) agent that monitors network states (such as queuing delay, observed throughput, etc.) and periodically modifies the congestion window (CWND) computed by its 'backbone' manually designed congestion controller, namely, TCP Cubic (see details in Section 3.1). Thus, fine-grained control is performed by TCP Cubic and coarse-grained 'corrections' are made by Orca's LCC.

We analyze the behavior of Orca for two desirable properties: (i) **Robustness:** Can Orca maintain high performance on traces similar to, but with slight differences from, those where it was trained on and has already excelled?; and (ii) **Performance:** Does the Orca controller utilize bandwidth well without significantly under- or over- subscribing?

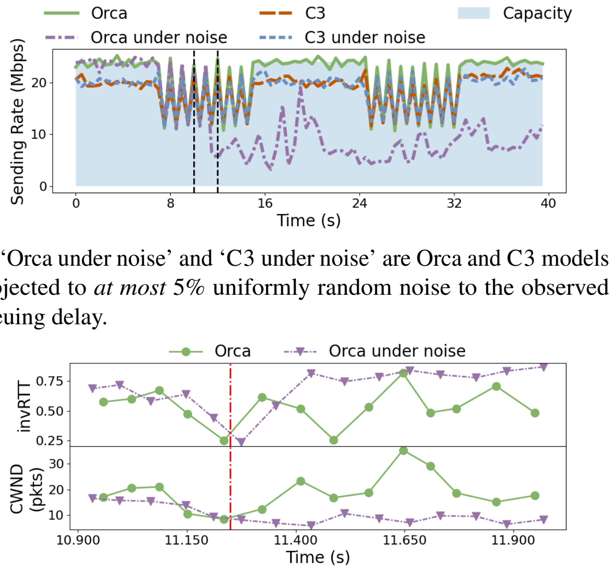
We comprehensively study several hand-constructed traces with frequent but controlled variations in the available bandwidth and distill our findings into two observations:

Observation #1: Orca is sensitive to perturbations. Consider the synthetic trace in Figure 1. The standard Orca model performs reasonably on this trace, mostly utilizing the available bandwidth even under bandwidth fluctuations. We then add random noise, uniformly sampled between -5% and 5%, to the observed delay state before passing it to the Orca LCC. Note that such small perturbations can manifest in practice due to measurement noise. Such measurement noise is common, e.g., due to reverse ACK congestion [38] and delayed ACKs [9], and granularity of throughput/drop measurements. The final state seen by the controller is shown as invRTT in Figure 1b. Figure 1b shows that even when the noisy input invRTT is high (e.g., right before 11.4s) implying low delays, the LCC continues to use a small CWND– leading to severe under-utilization of the link capacity. Thus, input points with similar features can lead to very different trajectories with Orca, showing that it is *not robust* to small amounts of noise.

Observation #2: Orca can enter bad states leading to poor utilization. Evaluating Orca on another synthetic trace (Figure 2a), we find that even without artificially added noise, Orca may enter into states with very low bandwidth utilization. To utilize the available bandwidth well, a controller should ideally meet the following property: increase (decrease) the CWND when experiencing low queueing (high) delays . Focusing on CWND increase behavior, Figure 2b shows that at the start of the misbehavior by Orca (around 17s – shown by the dashed vertical line), even though TCP suggests a large CWND to fill up the available capacity quickly, Orca instead forces a significantly lower CWND and worsens the behavior. Furthermore, the result suggests that once Orca enters such bad states, it may fail to recover from them. Similarly, Orca can also rapidly increase CWND when it should not, leading to persistent delays, loss, and poor throughput.

Such behaviors would not have occurred had the Orca LCC learner experienced these input points during training – if it were so, the learner would have seen a poor reward and "corrected" its action accordingly. However, LCCs cannot be expected to experience all possible values of inputs; in particular, Orca was only trained on a finite (albeit large) number of traces and thus may not have seen queuing delays for all possible input states. Such inevitable unexplored input

[1] While there are several RL-based controllers [2, 27, 50] for congestion control and our framework can work with any of them, we chose Orca owing to its available and reproducible codebase.



(a) ‘Orca under noise’ and ‘C3 under noise’ are Orca and C3 models subjected to *at most* 5% uniformly random noise to the observed queuing delay.

(b) invRTT represents the inverse normalized RTT ($minRTT/RTT$). High invRTT implies close to minimum RTT and hence should be followed by higher congestion windows in future time steps.

Figure 1: Orca under noises – (a) Sending Rate of Orca and C3 with and without noise. C3 is much more robust. (b) invRTT observed by Orca (including the noise) and its CWND, during the region within the dashed lines in (a). After the dashed vertical line, Orca under noise continues to use a small CWND while Orca, with a similar input, uses a higher CWND.

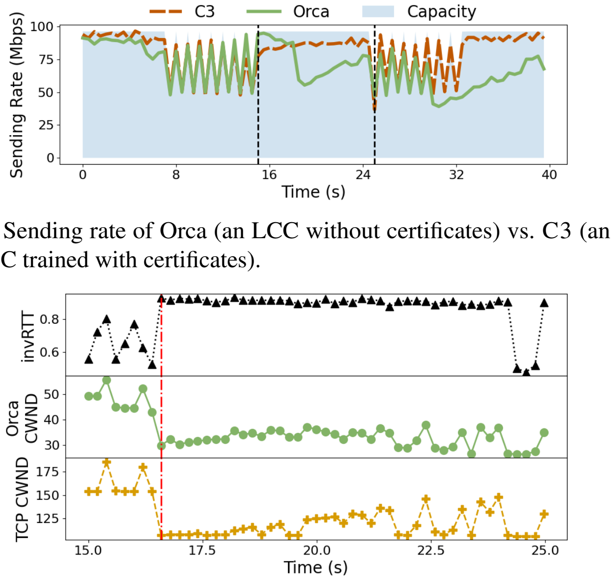
spaces cause unpredictable decisions at run-time, leading to bad worst-case behaviors.

2.2 C3: LCCs with Certificates

Our solution to the above issues is to incorporate *formal verification* of performance and robustness properties *into the LCC training loop*. The end outcome of our approach is illustrated in Figures 1a and 2a – the controllers C3 produces are more robust than Orca and can ensure good utilization.

Given a property, a *certificate* is a proof, produced by a formal verifier, that a learned model satisfies it for *all possible* inputs. For instance, for an LCC that takes the network state (including throughput, queuing delay, etc.) as the input and outputs the CWND to be used for the next time step, a desirable property may be of the form: “If queuing delay $> k \cdot RTT$ over a given period of time, the system should not increase the CWND.” A certificate then assures that for all possible network conditions matching the precondition (queuing delay $> k \cdot RTT$ for a given period of time), the model’s output matches the post-condition (does not increase the CWND).

The central idea in C3 is to compute such formal certificates via a verifier and use the certificate *during training* of a *base* learning-driven controller; in C3’s case, the base is Orca.



(a) Sending rate of Orca (an LCC without certificates) vs. C3 (an LCC trained with certificates).

(b) Trends of the observed invRTT ($= minRTT/RTT$), actual CWND action taken by Orca and the CWND suggested by TCP.

Figure 2: Orca entering critically bad states – (a) Sending rate of Orca suddenly drops and continues for a prolonged period of time while C3 maintains its sending rate. (b) Orca’s states for the region within the dashed lines in (a). With high invRTT (hence, low queueing delays), TCP increases the CWND (>100) but Orca continuously brings it back down (to <50).

At each training step[2], the verifier takes the current learned model (i.e., the Orca controller’s current weights) and the property of interest as inputs to compute a certificate.

Figure 3 presents an overview of training in C3. Here, given an input state range $[a,b]$, the verifier computes the interval $[p,q]$ overapproximating the set of outputs that the LCC can possibly produce. The verifier then checks whether the range of model outputs falls completely within the desired range $[l,u]$ defined in the property. We implement the verifier using abstract interpretation (Section 3.2).

Traditionally, formal verification is a problem with a boolean output: a system either satisfies or does not satisfy a property. However, since there are typically many more ways to fail a property than to pass it, a reward signal based on this bit is *sparse*, i.e., rarely positive. Learning from such sparse rewards is known to be hard [17]. Consequently, C3 computes an *interval distance* between the model output range $[p,q]$ computed by the verifier and the desired output range $[l,u]$. This quantity is used as *verifier feedback* (Section 4.4) within C3’s reward function, alongside the “raw” reward function that the underlying Orca LCC uses. While the raw feedback correlates with the empirical performance of the controller, the verifier feedback offers a smooth measure of how close

[2] Here, a step denotes one interaction of the LCC with the environment.

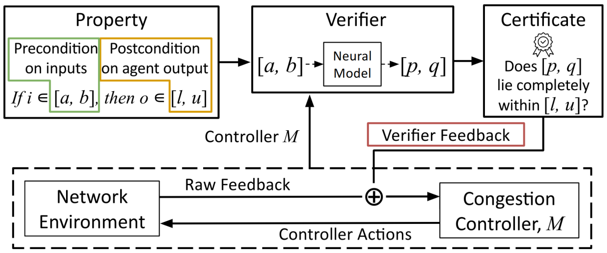


Figure 3: C3 Overview. Raw feedback is the training reward used by the base LCC, and verifier feedback is the additional signal provided in C3.

the current controller is to provably conforming to the property. Thus, the learner is nudged toward good average-case performance *and* worst-case property satisfaction.

We state a few salient features of the C3 framework:

First, congestion controller designers can plug in various properties of their choice into the framework and achieve a model fine-tuned for that property. In this paper, we discuss two properties, aligned with the notions of good performance and robustness for LCCs (Section 4.2). We discuss extensions to other properties in Section 8.

Second, because C3 decouples the baseline LCC from the property, congestion controller designers can use it with any LCC other than Orca and attain the benefits of the verifier feedback. Note that this implies that our verifier is also not tied to a specific learning algorithm (specifically, Actor-Critic RL used by Orca). Verifier feedback is provided alongside raw feedback, allowing the agent the autonomy to determine when and how to update its weights.

Third, our framework supports properties over partial states. For example, the Orca LCC takes multiple features in its input state, including queuing delay, throughput, loss rate, etc. However, one may only want to train using a property over queuing delay (for instance, the one mentioned above). For such properties, our abstract interpretation-based verifier uses symbolic values or intervals only for states of interest to the property (queuing delay in the above property), while using concrete values for other states.

3 Background

We devote this section to background on the Orca LCC and the technique of abstract interpretation and its application to neural models.

3.1 The Orca LCC

THR	The average throughput
l	The average loss rate of packets
DELAY	The average queuing delay of packets
n	The number of valid acknowledgement packets
m	The time between current report and the last report
$sRTT$	The smoothed RTT

Table 1: Observed network states in Orca.

Orca's RL formulation [2] uses a two-level control for the congestion window (CWND) - (i) fine-grained control that is performed by the TCP Cubic protocol, and (ii) coarse-grained control using a deep-RL agent. At periodic time-steps, Orca's RL agent monitors real-time feedback from the network environment (Table 1), including key metrics such as throughput and queuing delay – that it uses to compute a new CWND and enforces this CWND at every coarse-grained control step (aligned with the monitoring period). Thus, Orca modulates TCP's behavior through ML-based suggestions, as follows:

$$\text{CWND} = f_{\text{CWND}}(a, \text{CWND}^{\text{TCP}}) = 2^{2a} \times \text{CWND}_{\text{TCP}} \qquad (1)$$

where $a \in [-1.0, 1.0]$ is the output of Orca's coarse-grained control, and CWND_{TCP} is the current TCP suggested CWND. During training, Orca employs a heuristic reward function based on the Power metric [26], designed to increase as throughput rises or when loss rate and queuing delay decrease:

$$R_{\text{Orca}} = \frac{R}{R_{\max}} = \left(\frac{\text{THR} - \zeta \times l}{\text{DELAY}'}\right) / \left(\frac{\text{THR}_{\max}}{d_{\min}}\right) \qquad (2)$$

$$\text{DELAY}' = \begin{cases} d_{\min} & (d_{\min} \leq \text{DELAY} \leq \beta \times d_{\min}) \\ \text{DELAY} & \text{o.w.} \end{cases} \qquad (3)$$

Here ζ is a coefficient for controlling the impact of loss rate compared to the throughput, $\text{THR}_{\max}$ is the maximum throughput observed, and β is a coefficient > 1.

3.2 Abstract Interpretation

We build our certificates through *Abstract Interpretation* [11]. Here, one represents values — e.g., the system state, controller action, and reward — using symbolic representations, or *abstractions*, in a predefined language (the *abstract domain*). For example, we can set our *abstract states* to be hyperintervals with upper and lower bounds in each state space dimension. We denote abstract values with the superscript #. For a set of concrete states S, $\alpha(S)$ (known as the *abstraction function*) denotes the minimal-area abstract state containing S. For an abstract state $s^\#$, $\gamma(s^\#)$ (known as the *concretization function*) gives the set of concrete states represented by $s^\#$ (Figure 4).

The core of abstract interpretation is the propagation of abstract states $s^\#$ through a function $f(s)$ that captures system dynamics. For propagation, we assume that we have access to a map $f^\#(s^\#)$ that "lifts" f to abstract states (as shown in Figure 4). This function must satisfy the property $\gamma(f^\#(s^\#)) \supseteq \{f(s) : s \in \gamma(s^\#)\}$. Intuitively, $f^\#$ *overapproximates* the behavior of f: while the abstract state $f^\#(s^\#)$ may

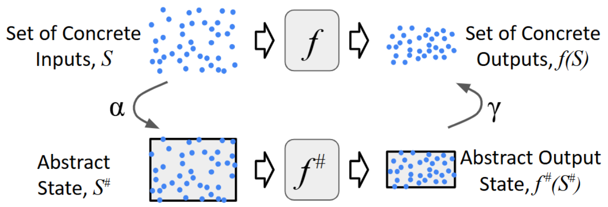

Set of Concrete Inputs, S
f
Set of Concrete Outputs, $f(S)$
α
γ
Abstract State, $S^{\#}$
$f^{\#}$
Abstract Output State, $f^{\#}(S^{\#})$

Figure 4: Abstract Interpretation *lifts* a concrete set of inputs to the abstract hyper-interval domain, and propagates the abstract state through the *lifted* $f^{\#}$ function.

include some states that are not reachable through the application of f to states encoded by $s^{\#}$, it will *at least* include every state that is reachable this way. For simplicity of notation, we will use the notation $f(x)$ to represent functions when x is concrete and their lifted counterparts when x is abstract. For example, $\pi(s^{\#})$ is short-hand for a function $\pi^{\#}(s^{\#})$ satisfying $\gamma(\pi^{\#}(s^{\#})) \supseteq \{\pi(s) : s \in \gamma(s^{\#})\}$ for every abstract state $s^{\#}$.

Abstract State Propagation. In C3, we utilize the box domain [36] for all the abstract states. In the box domain, for a state with m variables, its abstract counterpart in the domain is represented by an m-dimensional box. Each abstract state is a pair $s^{\#} = (b_c, b_e)_{\#}$, where $b_c \in \mathbb{R}^m$ is the center of the box and $b_e \in \mathbb{R}^m_{\geq 0}$ represents the non-negative deviations. The i-th dimension of the concretization, $\gamma(s^{\#})$, is given by interval $[(b_c)_i - (b_e)_i, (b_c)_i + (b_e)_i]$.

In other words, in the box domain, the abstract state $s^{\#} = (b_c, b_e)_{\#}$ represents all concrete states for which the i-th dimension is in the above interval.

We now showcase how the abstract state is propagated through the components of a neural network. When certifying a neural model, we need to propagate the input interval (Figure 3) through the model (denoted π), and any subsequent calculations (e.g., computation of f_{CWND}, defined in Equation (1)). This requires us to write abstract counterparts for the above functions. As an example, below, we explain how $s^{\#}$ is propagated for the basic matrix multiplication operation in neural networks. We describe propagation through other operations in Appendix A. For a matrix multiplication function f that multiplies the input $x \in \mathbb{R}^m$ by a fixed matrix $M \in \mathbb{R}^{m' \times m}$: $f(x) = M \cdot x$. The abstraction function of f is given by:

$$f^{\#}(s^{\#}) = (M \cdot b_c, |M| \cdot b_e)_{\#},$$

where $|M|$ is the element-wise absolute value of M.

4 C3

Now we present our approach. We first define our certified RL problem (Section 4.1), describe the properties C3 targets (Section 4.2) and the way our verifier constructs certificates (Section 4.3). Next, we present our mechanism for incorporating verifier feedback into training (Section 4.4). Finally, we give a way to reduce the constrained optimization problem in Section 4.1 into an unconstrained problem (Section 4.5).

4.1 Certified RL

RL Formulation. Following Orca, we formulate the certified learning problem in the general RL setting. For completeness and to aid the explanation of our approach, we provide more details that formalize the setting.

In RL, a learning task is modeled as a Markov Decision Process (MDP), defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, P, \mathcal{S}_0)$ where, $\mathcal{S}$ is a set of states; $\mathcal{A}$ is a set of actions; $\mathcal{S}_0$ is a distribution of initial states; $P(s' \mid s, a)$, for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, is a probabilistic transition function; and $r(s, a)$ for $s \in \mathcal{S}, a \in \mathcal{A}$ is a real-valued reward function. A *controller* (also called *policy*) in $\mathcal{M}$ is a possibly stochastic map π from states s to actions a[3]. A (finite) *trajectory* τ under π is a sequence $s_0, a_0, s_1, a_1, \ldots$ such that $s_0 \sim \mathcal{S}_0$, each $a_i = \pi(s_i)$, and each $s_{i+1} \sim P(s' \mid s_i, a_i)$. We denote by $R(\tau) = \sum_i \gamma^i r(s_i, a_i)$ the discounted sum of rewards along a trajectory τ, and by $R(\pi)$ the expected reward of trajectories unrolled under π.

For C3, R is the heuristic reward function described in Equation (2). We represent the state s_t at time t as the past k steps' observation of the network states $(\langle o_t, o_{t-1}, \ldots, o_{t-k} \rangle)$, a_t as the adjustments to CWND, the policy π as the congestion controller, and the transition as the network environment for a given link. Our observations o_i track the same set of monitoring statistics as in Orca (Table 1).

Certified RL. A *property* in our setting is a constraint of the form $\phi(\pi, X, Y)$, such that $\forall \langle s_i, \ldots, s_{i+k} \rangle \in X, a_{i+k} \in Y$, where $X \subseteq \mathcal{S}^k$ is the precondition and $Y \subseteq \mathcal{A}$ is the postcondition of ϕ respectively. Our goal in this paper is to learn controllers that *certifiably* satisfy such properties. That is, we expect our learning algorithm to produce, in addition to a controller π, a *certificate*, or *proof*, that $\phi(\pi, X, Y)$ holds. We write $\pi \vdash_c \phi$ if π satisfies ϕ and c is a certificate of this fact.

Our learning problem formalizes a setting in which the goal is to learn a controller that can *maximize* a standard heuristic reward while certifiably satisfying a property. The problem can be defined as

$$(\pi^*, c) = arg\,max_{\pi \in \Pi} \underset{\tau \sim (\mathcal{M}, \pi)}{\mathbb{E}} [R(\tau)], \text{s.t. } \pi \vdash_c \phi \qquad (4)$$

where R is the original reward function of the MDP $\mathcal{M}$.

This *constrained optimization problem* is difficult to solve - we present a tractable approach in Sections 4.4 and 4.5.

4.2 Properties in C3

The current implementation of C3 targets two classes of properties motivated by the observations from Section 2. Specifically, we consider a *performance property* that consists of two parts:

[3] In the implementation of C3, we only consider deterministic controllers.

1. If at step i, the past k-steps' normalized queuing delays are in the range $[p,\ 1.0]$, then $\text{CWND}_i - \text{CWND}_{i-1} \leq 0$.
2. If at step i, the past k-steps' normalized queuing delays are in the range $[0.0,\ q]$, $\text{CWND}_i - \text{CWND}_{i-1} \geq 0$.

Intuitively, these define the space of actions that a controller should take to utilize network bandwidth well. They assert lower and upper bounds on the change to CWND at each step under consistent large or small queuing delay observations. In congestion control, when the queuing delay is large – indicating the capacity may have been exceeded and potentially will lead to packet loss – we desire that the controller *does not increase* its CWND (reflected in #1 above). Likewise, a near-zero queuing delay indicates of the possibility that the capacity may not be fully used, where we desire the controller *does not decrease* its CWND (reflected in #2 above). We require our learned model to satisfy *both parts* of the above property.

We also consider a *robustness property* that requires that at step i, $\forall_{\eta \in [-\mu,\mu]} |\pi(s_i \times (1+\eta)) - \pi(s_i)| \leq \pi(s_i) \times \varepsilon$. In other words, adding small amounts of noise to the observed network state should not drastically change the value of CWND.

This property ensures that controller actions are robust to state perturbations arising from measurement errors (Section 2). Ensuring robustness at each time-step also lowers the likelihood that the slight perturbations that inevitably occur during a single step accumulate and snowball over future time steps and cause arbitrarily bad performance. Here, μ is a parameter for constraining the noise range on the observation and ε allows the controller's output to fluctuate within a small range.

4.3 Certificate Construction

Proceeding with our certified learning problem, the first key aspect of our method is the computation of certificates using abstract interpretation. Now we elaborate on this process.

An abstract state in C3 consists of k intervals over each state dimension of interest, with each interval tracking the value of the dimension over one of the past k time steps. Given a property $\phi(\pi, X, Y)$, we use intervals to initiate an abstract state $s_i^\# = \langle o_{i-k}^\#, \ldots, o_i^\# \rangle$ covering X for the past k steps. Then we soundly propragate this abstract state through the controller to obtain a symbolic output (action) representation $a_i^\# = \pi(s_i^\#)$. By comparing $a_i^\#$ with Y, we now obtain a proof of the property if $\gamma(a_i^\#) \subseteq Y$ (γ is the concretization function defined in Section 3.2). Thus, the indicator function $\mathbb{1}[\gamma(a_i^\#) \subseteq Y]$ is the *formal certificate* that a controller satisfies $\phi(\pi, X, Y)$.

As the neural controller in Orca predicts a *modification* to the value of CWND computed using TCP, our abstract state propagation includes the steps:

$$a_i^\# = \pi^\#(s_i^\#), \tag{5}$$

$$\text{CWND}_i^\# = f_{\text{CWND}}^\#(a_i^\#, \text{CWND}_i^{\text{TCP}}), \tag{6}$$

We now discuss how to construct certificates for the properties defined above.

Performance Certificate. In this case, at each step i, we keep track of CWND_{i-1}, the value of CWND at the previous monitoring step, and use this value to calculate the change to CWND. We define X as $\{s | \forall j \in [i-k, i], o_j.\text{DELAY} \in [p, 1.0]\}$ for the large-delay case, and as $\{s | \forall j \in [i-k, i], o_j.\text{DELAY} \in [0.0, q]\}$ in the small-delay case. Y equals $\{\text{CWND} | \text{CWND}_i - \text{CWND}_{i-1} \leq 0\}$ and $\{\text{CWND} | \text{CWND} - \text{CWND}_{i-1} \geq 0\}$ for the large-delay and small-delay constraints, respectively. Hence, our *performance certificate* is $\mathbb{1}[\Delta\text{CWND}_i^\# \subseteq Y]$, where $\Delta\text{CWND}_i^\# = \text{CWND}_i^\# - \text{CWND}_{i-1}$.

Robustness Certificate. For the robustness property, at each step i, we keep track of the current step's CWND_i from the controller given the observed states. For step i, X is $\{s | s_i \times (1+\eta), \eta \in [-\mu, \mu]\}$, $Y = \{\text{CWND} | abs(\text{CWND} - \text{CWND}_i)/\text{CWND}_i \leq \varepsilon\}$. Our *robustness certificate* is $\mathbb{1}[\text{CWNDCHANGE}_i^\# \subseteq Y]$, where $\text{CWNDCHANGE}_i^\# = (\text{CWND}_i^\# - \text{CWND}_i)/\text{CWND}_i$, representing the CWND change fraction.

4.4 Certificate Functions

Even with an efficient mechanism for certificate construction, our learning problem remains challenging for two reasons. First, property satisfaction is a sparse, discrete signal that is difficult to learn from. Second, Equation (4) requires us to balance the objective of learning for better average-case performance (represented by R) with the goal of satisfying the property on worst-case inputs. To address these challenges, we introduce a *certificate function* that *quantifies* property satisfaction and reduce the constrained problem into a tractable unconstrained optimization problem.

Our certificate function $C(\pi, c, \phi)$ is a smooth function which reaches its maximum $C_{\max}$ when $\pi \vdash_c \phi$. More precisely, $C(\pi, c, \phi) = \mathbb{E}_{\tau \sim (\mathcal{M}, \pi)}[R_{\text{verifier}}(\tau)]$, where $R_{\text{verifier}} = \sum_i \gamma^i r_{\text{verifier}}(s_i, a_i)$, for a reward function r_{verifier} derived from the verifier.

The verifier reward r_{verifier} computes the distance between the abstract output $\text{output}_i^\#$ computed by the verifier (recall that $\text{output}_i^\#$ is $\Delta\text{CWND}^\#$ and $\text{CWNDCHANGE}^\#$ for *performance* and *robustness* properties, respectively) and the postcondition Y_i asserted in the property. If $\text{output}_i^\#$ is already fully in Y_i, r_{verifier} is assigned 1.0, meaning we already have a good controller for this input condition.

More generally, the distance between a target area (e.g. $Y = [y_l, y_u]$) and the output (e.g. $\text{output}^\# = [o_l, o_u]$) is computed as:

$$D(Y, \text{output}^\#) = \begin{cases} 0, & \text{if } y_l > o_u \vee y_u < o_l \\ 1, & \text{if } y_l \leq o_l \wedge o_u \leq y_u \\ \frac{\text{VOLUME}(Y \cap \text{output}^\#)}{\text{VOLUME}(\text{output}^\#)}, & \text{other cases} \end{cases} \tag{7}$$

where VOLUME measures the volume of a set.

When a property has multiple constraints (e.g., our performance property has two constraints to be satisfied simultaneously.), we average the different D's:

$$r_{\text{verifier}}^{\text{performance}} = \frac{D(Y^{\text{large}}, \text{output}^{\text{large}}) + D(Y^{\text{small}}, \text{output}^{\text{small}})}{2} \quad (8)$$

where 'large' and 'small' are for large DELAY and small DELAY cases. In single constraint properties, $r_{\text{verifier}} = D$.

4.5 From Constrained to Unconstrained Learning

The introduction of the certificate function converts Equation (4) to

$$(\pi^*, c) = arg\,max_{\pi \in \Pi} \mathop{\mathbb{E}}_{\tau \sim (\mathcal{M}, \pi)} [R(\tau)], \text{s.t. } C(\pi, c, \phi) = C_{\max} \quad (9)$$

We can now use a known method [18, 34] to reduce this problem to an unconstrained problem. The idea here is to *convexify* the space Π of controllers by allowing stochastic combinations of controllers, which expands Π into its convex hull. Given the convexity of the resulting controller space, we can rewrite Equation (4) as

$$(\pi^*, c) = \min_{\hat{\lambda}} arg\,max_{\pi \in \hat{\pi}} \mathop{\mathbb{E}}_{\tau \sim (\mathcal{M}, \pi)} [R(\tau)] + \hat{\lambda}(C(\pi, c, \phi) - C_{\max}) \quad (10)$$

To optimally solve the above problem, we would need to perform an iteration over $\hat{\lambda}$. In practice, we simplify the problem further by assuming $\hat{\lambda}$ to be a hyperparameter controlling the trade-off between average-case performance and the certified performance, and omitting the convexification step. In addition, $C_{\max}$ is treated as a constant to represent certification satisfaction that can be omitted during optimization.

These simplifications leave us with the unconstrained optimization problem

$$(\pi^*, c) = arg\,max_{\pi \in \Pi} \mathop{\mathbb{E}}_{\tau \sim (\mathcal{M}, \pi)} [(1 - \lambda) R(\tau) + \lambda R_{\text{verifier}}(\tau)] \quad (11)$$

where $\lambda \in [0, 1]$ is the said hyperparameter. Intuitively, a small λ tends to learn the original controller (e.g., Orca) without any certified performance and $\lambda \to 1.0$ makes the learning fully guided by worst-case property adherence.

We note that Equation (11) has the form of a standard RL problem. We now use the underlying deep RL algorithm in Orca to solve this problem.

In practice, deep learning algorithms are seldom able to compute global optima; also, our reduction to unconstrained learning makes several simplifying assumptions as mentioned earlier. This means that while C3 *nudges* the learner in the direction of a fully certified controller, in practice, it will not necessarily arrive at a controller that provably satisfies a property *on all inputs*. However, as spelled out in Section 6, C3 can find controllers that provably satisfy performance and robustness properties on a *substantial fraction of the input space*, and significantly outperform baseline LCC approaches on the relative size of this space.

5 Implementation

Prototype. Our implementation is built on top of Orca. We modify the Linux Kernel (version 4.13) the same way as Orca for network state monitoring and for TCP Cubic to use the CWND computed by C3.

Verifier. We use Interval Bound Propagation (IBP) [21, 51] as our base verification algorithm to extract the abstract output. To support IBP, we wrap the neural controller with Sonnet [12], which is able to provide composable abstractions for each neural layer.

Since bound propagation introduces over-approximation, we cut the input space X into N symbolic components $\{X_i\}$ where $\cap_i X_i = \emptyset \wedge \cup_i X_i = X$ in each training step and propagate the small components through the computation components. A large number of components reduces the over-approximation and provides more accurate certificates but introduces computational overhead. After a thorough analysis (see Section 6.4), we use $N = 5$ for our training.

When representing the state input to the neural controller, we use the abstract representation for the variable of interest (e.g., queuing delay) and keep other variables as the observed values. In this way, we allow the freedom to explore different network conditions and collect real-time states to influence the certificate feedback but also keep track of the worst-case effect from the variable of interest.

Training. We build the C3 RL agent with Tensorflow [1]. Our neural controller architecture follows Orca: `FC-256` → `BN` → `Leaky ReLU` → `FC-256` → `BN` → `Leaky ReLU` → `FC-1` where `FC-N` means fully-connected feed-forward layer with output size `N`, `BN` is batch normalization. We use Python to implement the training-certification loop.

Orca's agent is built on top of the two delayed deep deterministic policy gradient algorithm (TD3) [19] to support continuous action learning effectively. Our framework follows the same agent learning algorithm.

Similar to Orca, we use a distributed learning architecture including multiple actors and one learner synchronizing the neural parameters from actors. Specifically, we use 256 actors, each interacting with a different network environment with stable bandwidth link during training. Each emulated network link uses a combination of characteristics in Table 2. For each actor, we pick the values for bandwidth and minimum RTT uniformly spaced within a range, and set the buffer size to be $2BDP$ for the given bandwidth and min. RTT. We emulate these links using Mahimahi [40]) and run a client-server pair on each link. We initiate a client request and let the server respond back with data continuously. We change the congestion controller on the server side and interface it with the actor.

6 Evaluation

We evaluate C3 against several benchmarks to answer the following questions:

Bandwidth	Minimum RTT	Buffer Size
[6$Mbps$ - 192$Mbps$]	[4ms - 400ms]	[3KB - 96MB]

Table 2: Training network environment characteristics.

Q1: Does C3 lead to better certified performance, compared to Orca? We define metrics to quantify this in Section 6.1.

Q2: Does C3 give good empirical performance?

Q3: How does C3 perform with various hyperparameter settings?

Q4: How does the training performance for C3 compare against Orca?

Q5: How much overhead is added by integrating the verifier in the training loop in C3?

We study two C3 models, trained with performance and robustness properties respectively, for the first two questions, and focus the rest on just the C3 model trained with performance property for brevity.

6.1 Experimental Setup

Baselines. We compare C3 against two types of baselines:
(i) Orca: We train Orca for multiple rounds of 50k epochs each (as specified in the original paper [2]). We checkpointed the model after each round, and selected the checkpoint with the best performance for our evaluations in this section.
(ii) TCP Variants: We also evaluate three TCP variants: Cubic [23], Vegas [6], and BBR [7]. Cubic treats packet loss as the main congestion signal and is used by Orca as its backbone congestion controller. Vegas instead uses delay as the main congestion signal. Finally, BBR combines observed delays and bandwidth estimation to make its decisions.

Traces. We evaluate C3 and the above baselines on 22 traces (see Appendix B for some samples), broadly of two categories:
(i) Synthetic Bandwidth Traces: We construct 18 traces with controlled, but sudden and frequent variations in the link capacity (similar to the traces shown in Section 2).
(ii) Real-world Traces: We also use four real-world traces from [47], including traces with highly variable bandwidths from commercial LTE networks such as AT&T, Verizon and TMobile.

Methodology. We train all models with eight servers[4] containing a total of 256 cores for our actor pool; servers are connected via 100G links and another identical node is used for the learner. During evaluation, we run a client that initiates a request to a server that responds by continuously sending data for the entire test duration – the congestion control scheme comes into play on the downlink from the server to the client. The link between the client and the server is emulated using Mahimahi [39]. We emulate each of the 22 traces (including synthetic and real-world traces) on the downlink and test each scheme back-to-back for each trace 5 times and report the average where applicable.

At each time step of the evaluation when the agent takes the state and produces an action, we also compute a certificate using the verifier with 50 components equally splitting the input space (see Section 5). We use a higher number of components than the training setup in order to precisely evaluate the certificate. We use such certificates during evaluation to compare C3's achieved property satisfaction against Orca's (more on that below).

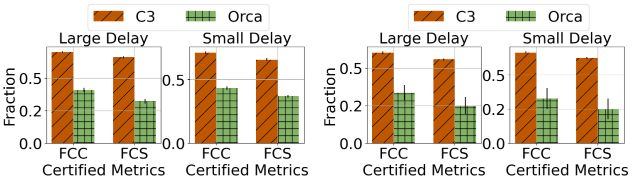
Evaluation Metrics. We use a variety of metrics to evaluate the certified and empirical performance of C3. The former is important because (as mentioned in Section 4.5) in practice, C3 may not necessarily arrive at a controller that provably satisfies a property *on all inputs*:
(i) Fraction of Certified Components (FCC): For each time step, FCC characterizes how many of the 50 components in the certificate satisfy the property being considered. We compute the average FCC over all time steps to obtain the FCC of a trace.
(ii) Fraction of Certified Steps (FCS): The FCS of a trace denotes the number of time steps in the trace where the computed certificate is satisfied by *all* 50 components. A low FCS value indicates bad worst-case performance – there were several time steps where the certificate was not fully satisfied and hence, worst-case behaviors may arise from the components that did not meet the property.
(iii) Average Utilization and Delay: The average utilization captures how well a congestion control scheme is able to utilize the available bandwidth while delays denote the congestion controller's ability to maintain bounded queues. Ideally, we want *high utilization and small, bounded delays*.

C3 Hyperparameters. We evaluate C3 over a range of parameter choices for λ and N. We choose the best performing C3 model ($\lambda = 0.25$, $N = 5$) for the results in Sections 6.2 and 6.3, and perform a detailed sensitivity analysis in Section 6.4. We focus on the performance property with parameters $(p = 0.75, q = 0.25)$ and robustness property with parameters $(\mu = 0.05, \varepsilon = 0.01)$. We follow Orca for all other hyperparameters in training.

6.2 Certified Performance

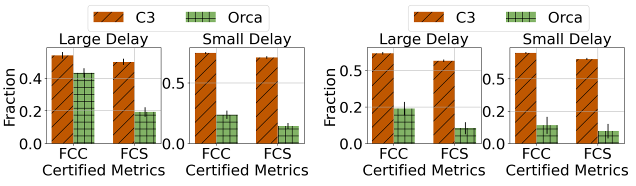
Note that achieving 100% satisfaction of the property is improbable in practice for the reasons mentioned in Section 4.5. We study closeness of C3-trained controllers to property satisfaction using the FCC and FCS metrics. We compare C3 against Orca as certified performance matters only for LCCs.

[4] Each server contains 32 Intel Xeon E5-2630 (2.40GHz) CPUs and 128GB of RAM.



(a) Synthetic Traces (b) Real World Traces

Figure 5: [Performance Property] Mean and std of FCC and FCS on different categories of traces – shallow buffer sizes (=1*BDP*).



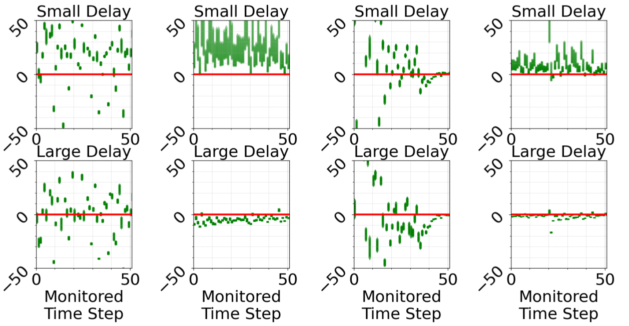
(a) Synthetic Traces (b) Real World Traces

Figure 6: [Performance Property] Mean and std of FCC and FCS on different categories of traces – large buffer sizes (=5*BDP*).

Performance Property. Figures 5 and 6 show the mean and standard deviation of FCC and FCS over all traces of a given type (synthetic/real), for two cases of the performance property (Section 4.2). C3 can achieve a high 0.54-0.69 FCC over all traces for the large delay case of the property; and 0.67-0.75 FCC for the small delay case of the property. Thus, on average, C3 satisfies the performance property for 50-70% of the components in the certificate. In contrast, Orca can only provide 0.24-0.44 FCC for the large delay case and 0.14-0.43 FCC for the small delay case. Thus, in practice, **C3 can provide significantly higher worst-case satisfaction of the performance property.**

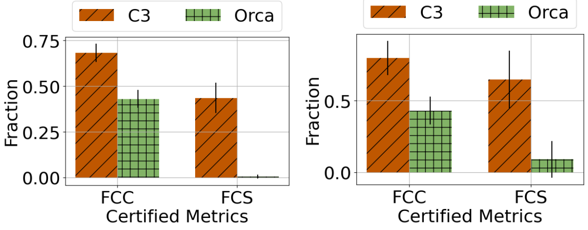
This observation is also reflected in the FCS numbers where C3 achieves an FCS of 0.50-0.66 (2.2-3.6× higher than Orca) for the large delay case of property and an FCS of 0.62-0.71 (1.8-6.5× higher than Orca) for the small delay case. Intuitively, this implies that C3 can provide a certificate with *full* satisfaction of the property on $>50\%$ of the time steps. In particular, C3 significantly outperforms Orca on real-world traces with large buffer sizes (see Figure 6b), where it can meet the property on 6.5× more time steps than Orca. Thus C3 can be expected to provide bounded delays in such cases (we will show this empirically below).

We provide a visual representation of what the FCC and FCS metrics imply in Figure 7 using two traces - for the depicted time slices, C3 has better bounds on the CWND action for most components at any time (reflecting the high FCC metric), and its output range satisfies the property fully on more time steps than Orca (reflecting the FCS metric).



(a) Trace 1: Orca (left), C3 (right) (b) Trace 2: Orca (left), C3 (right)

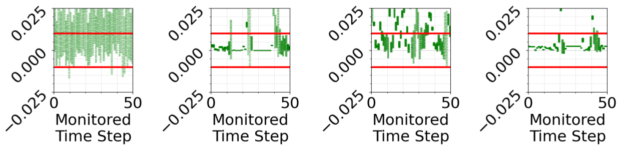
Figure 7: [Performance Property - y-axis: ΔCWND] Certified components distribution of Orca (top) and C3 (bottom) for 50 time steps on two traces. The figure shows the output bounds of the CWND action (ΔCWND) for each of the 50 components in the certificate (represented in colored). For the small delay case (top half in each trace), the property specifies $\Delta\text{CWND} \geq 0$, i.e., colored regions above the horizontal red line are desirable. For the large delay case (bottom half in each trace), the property specifies $\Delta\text{CWND} \leq 0$, i.e., colored regions below the horizontal red line are desirable.



(a) Synthetic Traces (b) Real World Traces

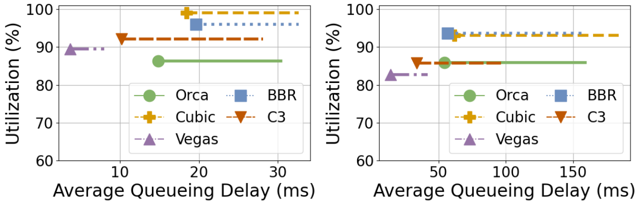
Figure 8: [Robustness Property] Mean and std of FCC and FCS on different categories of traces – 2*BDP* buffer sizes.

Robustness Property. Figure 8 presents the FCC and FCS results for the robustness property across two sets of traces. C3 achieves up to 0.81 FCC on real-world traces and 0.68 on synthetic traces, with FCS ranging from 0.44 to 0.70 – thus, a majority of components can formally satisfy the robustness property. Furthermore, most steps fully satisfy the property according to the FCS metric. In comparison, Orca, without certified learning, has poor worst-case behavior: with an FCS of less than 0.05, there are almost no steps where Orca fully maintains robustness when noise is introduced into the observed queuing delay. Figure 9 visually illustrates the CWNDCHANGE of two traces for Orca and C3 – C3 better bounds the CWND change to the target area.



(a) Trace 1: Orca (left), C3 (right) (b) Trace 2: Orca (left), C3 (right)

Figure 9: [Robustness Property - y-axis: CWNDCHANGE] Certified components distribution of Orca (left) and C3 (right) for 50 time steps on two traces. The figures show the output bounds of the CWND change fraction (CWNDCHANGE) for each of the 50 components in the certificate (represented in colored). The property specifies a target region for the CWND to fluctuate in, i.e., CWNDCHANGE within the horizontal red lines at $y = \pm 0.01$ is desirable.



(a) Synthetic Traces

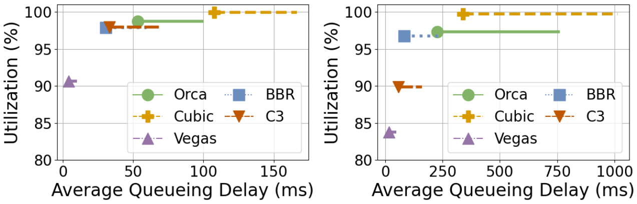
(b) Real World Traces

Figure 10: [Performance Property] Average throughput, normalized delay (icons) and p95 delay (end of lines) – small buffer sizes (=1*BDP*). Top-left indicates better overall performance.

6.3 Empirical Performance

To demonstrate the empirical performance of C3, we report three statistics averaged over all traces of a given type (synthetic/real): utilization, average delay, and average p95 delay. We first discuss the C3 controller trained with performance property and then the one trained with robustness property.

Performance Property. Figures 10 and 11 show the comparison for small buffer sizes (=1*BDP*) and large buffer sizes (=5*BDP*), respectively. Overall, **C3 consistently achieves smaller p95 delays compared to Orca**. C3's p95 delays are 7-38% and 29-78% smaller on shallow and large buffer links, respectively. Note that C3's performance property for large delays case prevents the CWND from increasing when delays become large (as shown in Section 6.2). Therefore training C3 to meet the performance property helps maintain low delays across all traces. At the same time, the small delay case of the performance property relies on a lower threshold for the queueing delay. Thus, C3 is trained to react when the queueing delay falls below this threshold. On highly variable bandwidth links with deep buffers (Figure 11b), this implies that C3 will only react when the queue has drained enough to meet the threshold – this leads to C3 experiencing 7.5% lower bandwidth utilization compared to Orca. Another crucial finding is that **C3 outperforms Orca on shallow buffer links.** Figures 10a and 10b show that C3 achieves 0.1% and 5% higher bandwidth utilization on shallow buffer links, while providing 31-38% smaller average delays. Note that Orca was trained on buffer sizes of 2*BDP*, thus it learned the optimal behavior for larger buffer sizes (evident by the high bandwidth utilization for all traces in Figure 11) but did not learn to react well to small buffer sizes. In particular, with small buffer sizes, more losses may occur and to prevent concomitant throughput drops, it is important to avoid losses *as soon as the delay increases*. In contrast, when trained to meet the performance property, C3 prevents CWND from increasing when delays become large. This reduces the number of packet losses by 18% and 30% on synthetic and real traces, respectively, compared to Orca, on shallow buffer links.



(a) Synthetic Traces

(b) Real World Traces

Figure 11: [Performance Property] Average throughput, normalized delay (icons) and p95 delay (end of lines) – large buffer sizes (=5*BDP*). Top-left indicates better overall performance.

Against TCP baselines, C3 always outperforms TCP Cubic in terms of delays and Vegas in terms of bandwidth utilization. Therefore, **C3 provides the best of both TCP Cubic and TCP Vegas.** Note that C3 uses TCP Cubic for its fine-grained control and its coarse-grained control is guided by a delay-based property (similar to what TCP Vegas proposes). In doing so, C3 prevents the bufferbloat issue of Cubic to a great extent – providing 13-47% and 57-84% smaller p95 delays for shallow and large buffer sizes, respectively. At the same time, it achieves 2-7% higher absolute utilization compared to TCP Vegas. Also, **C3 has smaller delays but slightly lower utilization compared to BBR**, as shown in Figures 10a, 10b and 11b, where C3 provides 12-37% smaller delays while achieving 0.92-0.98× bandwidth utilization.

Robustness Property. We apply a uniformly sampled noise within $[-5\%, 5\%]$ to the observed queueing delay for both Orca and C3. For all the synthetic and real-world traces, we measure the three metrics (utilization, average delay, and p95 delay) and we compute the percentage change in each of these metrics when noise is added (Figure 12). Overall, **Orca is unpredictable and highly variable in the face of minor noise while C3 trained with the robustness property is more robust.** While Orca can suffer up to 18% drop in utilization, C3 only sustains a maximum of 2% drop in its utilization

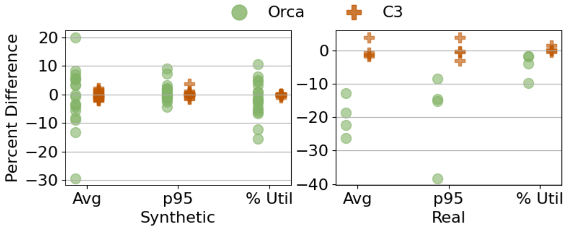


Figure 12: [Robustness Property] Percentage change in the average delay, p95 delay and average utilization (denoted Avg, p95 and % Util respectively) – when subjected to a 5% random noise. Closer to zero implies more robustness.

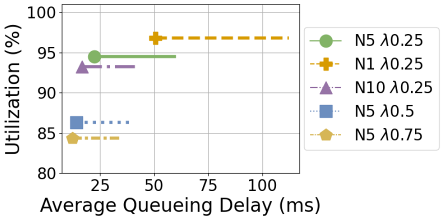


Figure 13: [Performance Property] Sensitivity of C3 to different values of the hyperparameters. N5 λ0.25 corresponds to the best-performing model used in Sections 6.2 and 6.3.

while still maintaining 95% bandwidth utilization on average. Note that adding noise may cause Orca to inadvertently make a better decision at some time steps in some traces where it was performing poorly earlier, thus leading to cases where adding noise to Orca may improve its performance.

6.4 Sensitivity Analysis

For brevity, we limit our analysis to the performance properties henceforth. Figure 13 shows the performance of C3 for various parameters choices. We explain the insights below.

Number of symbolic components. Note that we divide the input range into several components to improve the precision of certificates (Section 5). A larger number of symbolic components implies a more precise certificate (and hence, more accurate verifier feedback) because the input range is divided into finer slices, thereby resulting in less over-approximation. This is evident from Figure 13 where $N = 1$ yields loose certificates and is hence unable to bound delays leading to 1.88× higher p95 delays. On the other hand, $N = 10$ provides very accurate verifier feedback – which helps it to achieve 27% lower delays than $N = 5$ but in the process it loses out on bandwidth utilization. Further, $N = 10$ is also computationally expensive. Overall, $N = 5$ is found to be the best configuration balancing certificate precision and compute cost.

Weight of the verifier reward. By substituting different values for λ in Equation (11), one can increase or decrease the weight given to the verifier reward during learning. Figure 13 shows that increasing λ from 0.25 to 0.5 and 0.75 yields 32% and 42% smaller delays, respectively, but also results in 8.2% and 10.1% lower utilization as well. The reason is that more weight to the verifier reward guides the learner to high property satisfaction (and hence, more bounded delays) but is unable to optimize for the raw reward.

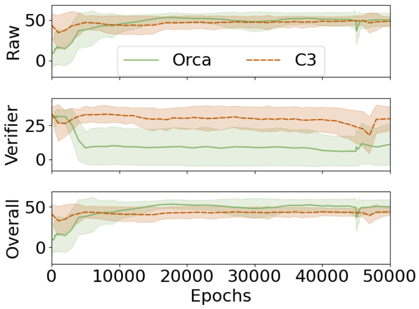


Figure 14: [Performance Property] Training Curves of Orca and C3.

6.5 Training Performance

We choose the best-performing C3 model trained on the performance property – $N = 5, \lambda = 0.25$ and compare the training performance for C3, against Orca. We train Orca using the same procedure and training parameters as specified in the original paper [2], picking the best checkpoint when trained on multiple training rounds. We measure the raw reward, verifier reward, and the overall reward (as in Equation (11)) at each epoch and show the resulting training curve in Figure 14. As the training progresses, Orca's raw reward increases as expected. However, its verifier reward drops. This indicates that just the raw reward is not enough to ensure high satisfaction of the property – in fact, the decreasing verifier reward over epochs suggests that the Orca learner optimizes for the raw reward in a way that can reduce the satisfaction of the property! In C3, we can gain a better verifier reward, without significantly sacrificing the raw reward.

6.6 Training Overhead

Orca	C3 - Performance Property		
	N=1	N=5	N=10
29.6	17.7	6.2	3.4

Table 3: Epoch rates.

Unavoidably, verification introduces additional overhead. We measure the mean epoch rate (epochs per second) across 256 actors, as shown in Table 3. We vary the number of components (N) in the certificate (Section 5) to obtain the results. Each additional component requires another pass through the CWND$^{\#}$ computation. In the performance property, we account for two delay constraints (Section 4.2), thus the verifier is invoked twice for each component. Therefore, the per epoch computational complexity is $O(\text{C3}_{\text{Performance}}) = (2N) \cdot O(\text{Verifier}) + O(\text{Orca})$. This results in the increased time for each epoch and reduced training throughput with increasing N.

7 Related Work

Learning-based Congestion Controllers. Several ML-based approaches have been proposed for congestion control. In addition to Orca [2], [45] utilizes imitation learning, [27] introduces a deep RL-based controller, [15] focuses on online learning, and [46] applies offline optimization. Additionally, [50] explores data-driven congestion control design. However, none of these LCCs offer certified guarantees after training, and their neural networks can exhibit poor and unpredictable worst-case behavior similar to Orca's.

Verification for ML-based systems. Various machine learning techniques have been proposed for computer systems problems [30,35], including resource allocation [37], memory access prediction [25], offline storage configuration recommendation [32], database query optimization [33], and storage placement optimization [24,52]. Formal verification is a powerful tool for assessing the worst-case performance of such systems, as explored, for example, in [13,31]. Similarly, [16] verifies LCCs. All these approaches offer binary post-training verification feedback.

A recent system [43] leverages verification tools during training to generate high-quality counterexamples to augment the training dataset. This approach produces a binary certificate at the end of training; also, like the above schemes, there is no certified feedback during training. In addition, [43]'s technique to identify input spaces to look for counterexamples applies only to supervised learning algorithms.

Certified learning. The concept of certified learning traces back to NaVer [41] and has since evolved with works like [8] (which synthesizes program following constraints), [28,29] (which leverage temporal logic formula to shape the RL reward), [49] (which integrates verification into the training loop by making it differentiable), [48] (which includes verification with model-based RL), and [4] (which adds a learned shield to safeguard learning performance).

Unlike C3, these techniques were not aimed at the systems domain which entails considering (1) systems properties of interest, specifying and modeling them, constructing their certificates, and integrating them with system performance-related reward functions, (2) systematically using abstract interpretation, balancing over-approximation and computational cost of training when exploring systems input spaces, and (3) implementing and evaluating the certified performance of the resulting systems in practical settings.

8 Concluding Remarks and Discussion

We have presented C3, the first LCC approach in which the learned controller comes with a certificate of satisfying robustness or performance properties. We have demonstrated the need for such certificates by highlighting certain key flaws in current LCCs. We have given a method for incorporating formal certificates into the training process. We have shown that this approach offers significantly better certified performance without sacrificing much in terms of average-case performance. We hope that our methods and results will inspire further research on ways to regulate learning-based systems. Now we identify a few concrete directions for such research.

Beyond Congestion Control. Fundamentally, C3 builds on top of a backbone learning-based system and a given property on the input-output behavior of the learned components. This means that it is not restricted to Orca or congestion control systems. Irrespective of the learning algorithm used by the backbone (reinforcement learning – e.g., actor-critic, Q-learning, etc. – or supervised learning), one can construct loss functions using the same framework to guide "certificate-in-the-loop" training. Developing C3-like approaches to other learned systems is an interesting direction of future work.

Richer Properties. C3 is not limited to the performance and robustness properties in the paper. Any property following the form $\phi(\pi, X, Y)$ can be supported right out-of-the-box. However, C3 does not currently support temporal properties. Temporal properties regulate sequences of actions, which requires considering the influence of the environment, and thus, a transition representation for the environment. (In general, such a representation requires strong assumptions about the environment.) Augmenting C3 with hand-designed or learned models of the environment is an interesting avenue for future exploration.

Increasing the Precision of Verification. The certificates C3 computes are sound but incomplete due to over-approximation in abstract interpretation. In particular, the box domain in our case may include points not in the original input set (see Figure 4). This means that a neural controller violating the certificate may not necessarily violate the property itself. However, any controller satisfying the certificate is guaranteed to satisfy the property. Over-approximation inaccuracy can be mitigated in several ways: (i) Using more complex domains: Polyhedral domains, for example, reduce over-approximation but come with higher computational overhead. Given the inclusion of certificates in training, we opted for the lightweight box domain in our work. (ii) Domain subdivision: Our current work divides the domain into smaller components to improve precision (Section 5). Future work could incorporate smart sampling techniques to focus on components likely to lead to edge cases, thus reducing computational costs.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 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*, pages 632–647, 2020.

[3] Anup Agarwal, Venkat Arun, Devdeep Ray, Ruben Martins, and Srinivasan Seshan. Towards provably performant congestion control. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 951–978, Santa Clara, CA, April 2024. USENIX Association.

[4] Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. *Advances in neural information processing systems*, 33:6172–6183, 2020.

[5] Venkat Arun, Mina Tahmasbi Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. Toward formally verifying congestion control behavior. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 1–16, New York, NY, USA, 2021. Association for Computing Machinery.

[6] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: new techniques for congestion detection and avoidance. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, page 24–35, New York, NY, USA, 1994. Association for Computing Machinery.

[7] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.

[8] Swarat Chaudhuri, Martin Clochard, and Armando Solar-Lezama. Bridging boolean and quantitative synthesis using smoothed proof search. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 207–220, 2014.

[9] Xiaoqi Chen, Hyojoon Kim, Javed M. Aman, Willie Chang, Mack Lee, and Jennifer Rexford. Measuring TCP round-trip time in the data plane. In Ang Chen and Laurent Vanbever, editors, *Proceedings of the 2020 ACM SIGCOMM 2020 Workshop on Secure Programmable Network Infrastructure, SPIN@SIGCOMM 2020, Virtual Event, USA, August 14, 2020*, pages 35–41. ACM, 2020.

[10] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.

[11] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[12] DeepMind. Sonnet: A library for constructing neural networks. https://github.com/google-deepmind/sonnet, 2024. Accessed: September 15, 2024.

[13] Arnaud Dethise, Marco Canini, and Nina Narodytska. Analyzing learning-based networked systems with formal verification. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[14] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-Learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, Renton, WA, April 2018. USENIX Association.

[15] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. {PCC} vivace:{Online-Learning} congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, 2018.

[16] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 305–318, 2021.

[17] Jonas Eschmann. Reward function design in reinforcement learning. *Reinforcement Learning Algorithms: Analysis and Applications*, pages 25–33, 2021.

[18] Yoav Freund and Robert E Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.

[19] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[20] Saksham Goel, Benjamin Mikek, Jehad Aly, Venkat Arun, Ahmed Saeed, and Aditya Akella. A performance verification methodology for resource allocation heuristics, 2024.

[21] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[22] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[23] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[24] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S Gunawi. {LinnOS}: Predictability on unpredictable flash storage with a light neural network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 173–190, 2020.

[25] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. In *International Conference on Machine Learning*, pages 1919–1928. PMLR, 2018.

[26] J. Jaffe. Flow control power is nondecentralizable. *IEEE Transactions on Communications*, 29(9):1301–1306, 1981.

[27] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[28] Yuqian Jiang, Suda Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. Temporal-logic-based reward shaping for continuing reinforcement learning tasks. In *Proceedings of the AAAI Conference on artificial Intelligence*, volume 35, pages 7995–8003, 2021.

[29] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

[30] Marios Evangelos Kanakis, Ramin Khalili, and Lin Wang. Machine learning for computer systems and networking: A survey. *ACM Computing Surveys*, 55(4):1–36, 2022.

[31] Guy Katz. Augmenting deep neural networks with scenario-based guard rules. In *Model-Driven Engineering and Software Development: 8th International Conference, MODELSWARD 2020, Valletta, Malta, February 25–27, 2020, Revised Selected Papers 8*, pages 147–172. Springer, 2021.

[32] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Selecta: Heterogeneous cloud storage configuration for data analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 759–773, 2018.

[33] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H Chi, Jialin Ding, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. Sagedb: A learned database system. *CIDR 2019 - 9th Biennial Conference on Innovative Data Systems Research*, 2021.

[34] Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR, 2019.

[35] Martin Maas. A taxonomy of ml for systems problems. *IEEE Micro*, 40(5):8–16, 2020.

[36] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3578–3586. PMLR, 2018.

[37] Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices*, 53(2):184–198, 2018.

[38] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily R. Blem, Hassan M. G. Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. TIMELY: rtt-based congestion control for the datacenter. In Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye, editors, *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pages 537–550. ACM, 2015.

[39] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, Santa Clara, CA, July 2015. USENIX Association.

[40] Ravi Netravali, Anirudh Sivaraman, Keith Winstein, Somak Das, Ameesh Goyal, and Hari Balakrishnan. Mahimahi: A lightweight toolkit for reproducible web measurement. *ACM SIGCOMM Computer Communication Review*, 44(4):129–130, 2014.

[41] Luca Pulina and Armando Tacchella. Never: a tool for artificial neural networks verification. *Annals of Mathematics and Artificial Intelligence*, 62:403–425, 2011.

[42] Simon Scherrer, Markus Legner, Adrian Perrig, and Stefan Schmid. Model-based insights on the performance, fairness, and stability of bbr. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 519–537, New York, NY, USA, 2022. Association for Computing Machinery.

[43] Cheng Tan, Changliu Liu, Zhihao Jia, and Tianhao Wei. Building verified neural networks for computer systems with ouroboros. *Proceedings of Machine Learning and Systems*, 5:728–742, 2023.

[44] Ranysha Ware, Adithya Abraham Philip, Nicholas Hungria, Yash Kothari, Justine Sherry, and Srinivasan Seshan. Ccanalyzer: An efficient and nearly-passive congestion control classifier. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 181–196, New York, NY, USA, 2024. Association for Computing Machinery.

[45] Wenting Wei, Huaxi Gu, and Baochun Li. Congestion control: A renaissance with machine learning. *IEEE network*, 35(4):262–269, 2021.

[46] Keith Winstein and Hari Balakrishnan. Tcp ex machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review*, 43(4):123–134, 2013.

[47] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 459–471, 2013.

[48] Chenxi Yang, Greg Anderson, and Swarat Chaudhuri. Certifiably robust reinforcement learning through model-based abstract interpretation. In *2024 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 233–251. IEEE, 2024.

[49] Chenxi Yang and Swarat Chaudhuri. Safe neurosymbolic learning with differentiable symbolic execution. In *International Conference on Learning Representations*, 2022.

[50] Chen-Yu Yen, Soheil Abbasloo, and H. Jonathan Chao. Computers can learn from the heuristic designs and master internet congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 255–274, New York, NY, USA, 2023. Association for Computing Machinery.

[51] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*, 2019.

[52] Giulio Zhou and Martin Maas. Learning on distributed traces for data center storage systems. *Proceedings of Machine Learning and Systems*, 3:350–364, 2021.

Appendix

A Abstract State Propagation

We present the abstract state propagation on more operations involved in the program and neural network certification below.

Add. The 'Add' concrete function f replaces the i-th element in the input vector $x \in \mathbb{R}^m$ with the sum of the j-th and k-th element:

$$f(x) = (x_1, \ldots, x_{i-1}, x_j + x_k, x_{i+1}, \ldots x_m)^T.$$

The abstraction function of f is given by:

$$f^\#(s^\#) = (M \cdot b_c, M \cdot b_e)_\#$$

where $M \in \mathbb{R}^{m \times m}$ is a matrix that allows $M \cdot b_c$ to replace the i-th element of $s^\#$ with the sum of the j-th and k-th element.

ReLU. For a concrete element-wise ReLU operation over $x \in \mathbb{R}^m$:

$$\text{ReLU}(x) = (\max(x_1, 0), \ldots, \max(x_m, 0))^T,$$

the abstraction function of ReLU is given by:

$$\text{ReLU}^\#(s^\#) = \Bigg(\frac{\text{ReLU}(b_c + b_e) + \text{ReLU}(b_c - b_e)}{2}, \frac{\text{ReLU}(b_c + b_e) - \text{ReLU}(b_c - b_e)}{2}\Bigg)_\#.$$

where $b_c + b_e$ and $b_c - b_e$ denotes the element-wise sum and element-wise subtraction between b_c and b_e.

B Samples of Synthetic and Real-World Traces

We show samples of the kinds of traces used for our evaluation. Figures 15 to 17 show the three types of synthetic traces used in the evaluation. We construct 18 traces of these forms in total and use them for the results in Sections 2 and 6. Figures 18 and 19 show samples of real-world traces from commercial LTE providers that we use in Section 6.

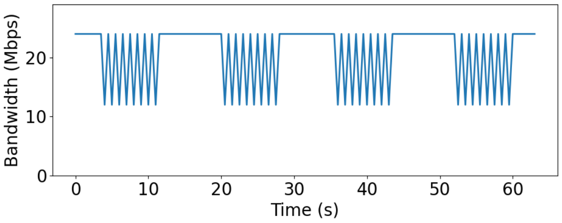


Figure 15: Fluctuating bandwidths at regular intervals.

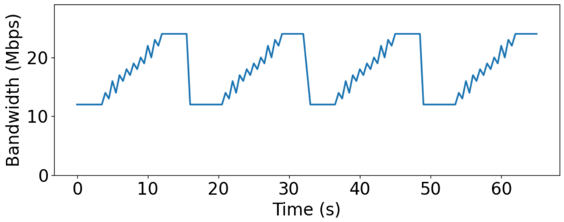


Figure 16: Gradually increasing bandwidth followed by sudden drop.

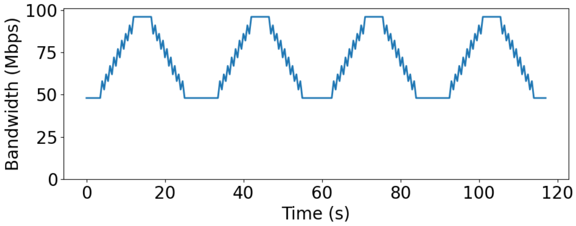


Figure 17: Gradually increasing and gradually decreasing bandwidths.

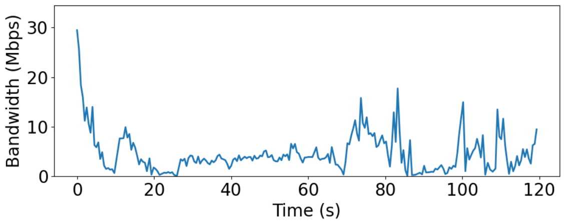


Figure 18: Highly variable bandwidth link from a commercial LTE provider.

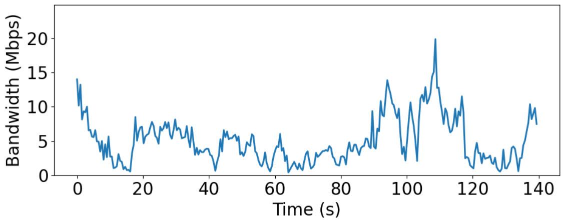


Figure 19: Highly variable bandwidth link from a commercial LTE provider.