The Case for Model-Driven Interpretability of Delay-based Congestion Control Protocols

Muhammad Khan
New York University Abu Dhabi, UAE
mk7406@nyu.edu

Yasir Zaki
New York University Abu Dhabi, UAE
yasir.zaki@nyu.edu

Shiva Iyer
New York University, USA
shiva.iyer@nyu.edu

Talal Ahamd
Google, USA
ahmad@cs.nyu.edu

Thomas Poetsch
New York University Abu Dhabi, UAE
thomas.poetsch@nyu.edu

Jay Chen
ICSI Berkeley, USA
jay.chen@nyu.edu

Anirudh Sivaraman
New York University, USA
anirudh@cs.nyu.edu

Lakshmi Subramanian
New York University, USA
lakshmi@nyu.edu

ABSTRACT

Several new delay-based congestion control protocols are proposed with complex non-linear control loops. However, analyzing and interpreting their exact behavior is exceptionally hard, especially when the underlying network is highly variable, like in cellular networks. This paper proposes Model-Driven Interpretability (MDI), a new congestion control modeling framework to reason about the behavior of such delay based protocols under variable network conditions. The MDI framework allows us to derive a model version of a delay-based protocol by simplifying a congestion control protocol's response into a guided random walk over a two-dimensional Markov model. We demonstrate the case for the MDI modeling framework by showing how one can use MDI to analyze and interpret the behavior of two sample delay based protocols over cellular channels: Verus and Copa. Our results show that the model versions of the protocols successfully approximate the throughput and delay characteristics of these protocols across variable network conditions. We show that the learned model of a protocol can provide key insights into the convergence properties of an algorithm and can potentially be useful as a drop-in replacement of the original protocols.

1 INTRODUCTION

Cellular channels are known to fluctuate rapidly over short periods of time [29]. 3G and LTE network measurements [10, 11, 16] demonstrated that variations in the channel cause significant performance differences across carriers, access technologies, geographic regions, and time. Rapid channel fluctuations cause loss-based congestion control (CC) algorithms to overreact and under perform [3], resulting in buffer-bloat and high delays [7, 8, 13]. Several protocols such as Sprout [27], Copa [2], Verus [29] and BBR [3] have demonstrated significant performance gains against traditional TCP variants over highly variable network channels. A common recurring theme across these protocols is to use *delay-based signals* to measure the network congestion state. While there is a broad array of research on the dynamics of loss-based CC protocols [4, 18, 23], we still lack a principled framework for understanding the dynamics of delay-based protocols.

This paper proposes *Model-Driven Interpretable (MDI)* CC framework, aiming to enhance the ability to interpret delay-based CC protocols' behavior. Given any protocol, the MDI framework uses empirical data on the protocol's performance for training a stochastic two-dimensional discrete-time Markov model to represent the protocol's behavior. In essence, using the empirical behavior of a protocol across diverse network conditions, MDI converts a protocol into a stochastic random walk in Markovian state space. Each state transition is determined by the delay variation feedback from the network. The protocol MDI aims to:

(1) Closely approximates the mean/variance of the throughput and delay distributions of the original protocol.
(2) Track the original protocol's temporal behavior, i.e., how to react to variations in network conditions.

We note that achieving these two properties for a broad array of protocols is a non-trivial task. In the MDI framework, the notion of protocol memory is implicitly captured in the definition of the state space (transition probabilities), and the stochastic random walk using delay feedback. While the state space does represent a significant approximation to the original protocol, we show that in practice, MDI successfully approximates the behavior of the original protocol.

To evaluate MDI, we developed MDI versions of two different protocols: Verus [29] and Copa [2]. Using real-world cellular traces in 3G and 4G networks and across synthetic highly variable network conditions, we show that the MDI version of a protocol closely approximates the throughput and delay distributions of the original protocol and temporally tracks the protocols' behavior. We demonstrate three specific benefits of MDI in this paper:

Visualizing Protocols: A state-space representation of a protocol enables visual understanding of its behavior including measuring how state-space transitions vary across: (i) protocols under the same network condition; (ii) network conditions under the same protocol.

Reasoning about Convergence: By representing a protocol in a Markovian state space, one can derive the mixing time and the corresponding stationary distribution of the MDI version of a protocol which we show empirically to closely mirror the measured statistical properties of the protocol.

Drop-in replacement: MDI models can be used as a drop-in replacement for the original protocol to potentially reduce the implementation complexity of new protocols.

The MDI framework as presented in this paper is a smaller part of a much larger puzzle of understanding the properties of delay-based control protocols. What this paper has primarily shown is the feasibility of the MDI framework in modeling two such protocols using a Markov Model representation. One long term motivation of using a Markovian framework is to leverage the vast body of statistics literature on Markov models and random walks to understand the stability, dynamics, and adaptivity of delay-based protocols. While we have shown initial empirical evidence for analyzing convergence properties of protocols and visualizing protocols using MDI, a detailed statistical analysis of protocols using this framework is a subject for future work and is beyond the scope of this paper.

2 MDI DESIGN

The main idea of MDI is to build a model that reflects the statistical properties, providing a more intuitive and predictable understanding of the protocol behavior. At an abstract level, MDI assumes that CC protocols can be modeled by the relationship between the current and the next state, where each state is a tuple of the relative change in the network delay and the sending window size.

2.1 Modeling Delay based Control

Consider a protocol P that uses delay-variations as a congestion signal. One can imagine such a protocol maintains a recent history of delay observations, which can be used to estimates the next sending window or rate. Let us consider an epoch as the unit of time for making a decision, which can be a variable or a fixed period depending on the protocol.

The challenge in a Markov model representation of a protocol P is determining the appropriate state space and mapping the protocol actions to transitions within the states. The most straightforward approach is to map the absolute values directly by describing a state as (d_i,w_i) where d_i and w_i are the experienced delay and sending window in an epoch i, respectively. For brevity, we use d and w (without the epoch subscript i) to abstractly represent the observed delay and window parameters. While a two-variable state space using (d, w) is simple, it may not be rich/generic since it may not be sufficient to capture the variations in these parameters. If one were to represent the state space using a history of delay and window measurements, the state space representation could be much more vibrant, but correspondingly much harder to accurately learn. In fact, for each additional dimension in the state space, we need an order of magnitude more training data to determine the state transitions. To capture the variations of the delay and window in the state space, we also consider: (1) relative change in the delay across neighboring epochs (captured by $\alpha(d)$); (2) relative change in the window across adjacent epochs (obtained by $\beta(w)$). These four parameters provide a richer representation of the state space. However, the training data required for the 4-dimensional space is at least two orders of magnitude more than the (d, w) space. To balance between state complexity and state richness, we chose to condense these four parameters into two composite parameters as $\alpha(d) \cdot log_{10}(d)$ and $\beta(w) \cdot log_{10}(w)$. By representing the delay and window in log space and quantizing the values (described in Section 2.2), we are able to better delineate variations in relative delay (or window) changes in comparison to variations in the actual delay (or window) values across different buckets in the state space. The quantization of these values also helps in maintaining a condensed two parameter representation of the four parameters: window, delay, relative change in window size and the relative change in delay across epochs. We do note that one can choose alternate state space representations for the MDI framework; the key requirements are to balance between the number of quantized states in the state space with the ability to capture protocol dynamics across different network conditions.

2.2 Discrete-time Markov Model States

A discrete-time Markov model of a protocol is represented in the form of a state-transition probability matrix. The matrix describes transition probabilities from one state to another obtained by training a protocol on a large set of network configurations. We call this the *training phase* of the Markov model. To capture the protocol behavior, the matrix should include as many states as the ones observed during the training. The state is defined as a tuple with value pairs of $(\hat{d_i}, \hat{w_i})$. Where $\hat{d_i}$ and $\hat{w_i}$ are calculated using the current epoch's packet delays (d_i) and sending-window (w_i) and the previous epoch's delay (d_{i-1}) and sending-window (w_{i-1}):

$$\hat{d_i} = \left[\left(\frac{d_i}{d_{i-1}}\right) - 1\right] * log_{10}(d_i) \quad (1)$$

$$\hat{w_i} = \left[\left(\frac{w_i}{w_{i-1}}\right) - 1\right] * log_{10}(w_i) \quad (2)$$

Assume that a protocol P adjusts the congestion window as a function of delay feedback. A user executing protocol P has currently the following values: the current sending window w_i, and the previous epoch delay feedback d_{i-1}. To decide on the value of the next window w_{i+1}, the user has to first identify the current delay d_i. The protocol P decides the next window w_{i+1} based on the following factors: the prior window w_i, and the delay variations. Only upon observing d_i, P would be aware of the true represented state $(\hat{d_i}, \hat{w_i})$ in the model space of the protocol. Essentially, given an initial window w_i and delay d_{i-1}, the protocol P has three variations that influence a transition from $(\hat{d_i}, \hat{w_i})$ to $(\hat{d_{i+1}}, \hat{w_{i+1}})$: (i) the variation in the initial observation d_i; (ii) the variation in the decision making of w_{i+1}; (iii) the variation in the next delay observation d_{i+1}. Note that, it is not necessary for two users running the same protocol P and in the same state $(\hat{d_i}, \hat{w_i})$, to derive the same next window w_{i+1}. This decision is influenced by two factors: (i) different windows/delays values could effectively arrive at the same model state $(\hat{d_i}, \hat{w_i})$; (ii) different flows may observe variations in prior observations of delays and windows.

2.3 Deriving the MDI Transition Matrix

The key assumption that MDI makes is that the state transition from $(\hat{d_i}, \hat{w_i})$ to $(\hat{d_{i+1}}, \hat{w_{i+1}})$ can be captured by a guided Markov model with two basic properties: the delay feedback guides the direction of the window change (increase or decrease), and the delay variations of d_i and d_{i+1} have an inherent randomness that influence the protocol choice of the next window w_{i+1}. The guided Markov assumption is clearly an approximation of the original protocol behavior.

To derive the transition matrix, we use a protocol emulation strategy in a constrained network environment. Consider a network simulation environment where one can execute the protocol P under

various network conditions and background traffic. Our setup's network environment is defined by a set of network traces that specify bandwidth, packet loss, and RTT variations. The protocol P can be executed by simulating network flows executing the protocol in the presence of competing traffic. We perform a broad array of network simulations by varying the network traces and the background traffic emulating several real-world protocols, including P. For each simulation, we measure the state transitions of P across the model states. By observing all possible state transitions of $(\hat{d}_i, \hat{w}_i)$, with $\hat{d}_i$ ranging from $\hat{d}_{min}$ to $\hat{d}_{max}$, and $\hat{w}_i$ ranging from $\hat{w}_{min}$ to $\hat{w}_{max}$, a 2D Markov chain is created defining the following states: current state $(\hat{d}_k, \hat{w}_l)$ and next state $(\hat{d}_r, \hat{w}_v)$, where k and l are the current state indexes of $\hat{d}_i$ and $\hat{w}_i$, respectively. Similarly, r and v represents the next state indexes. To reduce the state space of possible values for $(\hat{d}_i, \hat{w}_i)$, we quantize these values into small buckets. MDI captures the state transitions in the form of a transition probability matrix written as:

$$p_{(k,l),(r,v)} = p[(\hat{d}_k, \hat{w}_l)|(\hat{d}_r, \hat{w}_v)]. \quad (3)$$

Thus, $(\hat{d}_k, \hat{w}_l)$ defines a specific row in the transition matrix. Depending on $\hat{d_{i+1}}$ next value, represented by r, we obtain a subset of values from this specific row (i.e., the probability going to any of the possible $\hat{w_{i+1}}$ in the state $(\hat{d}_r, \hat{w}_v)$.

2.4 Model Training Methodology

This paper focuses on training two delay-based protocols: Verus and Copa. The training is performed over a large sample of cellular traces covering a wide range of diverse scenarios. We ran each protocol through a network emulator over a large set of traces randomly synthesized from the training traces. In each run, the protocol behavior is captured by logging the set of congestion windows and their experienced correlated delays. Next, the logged window and delay values are quantized (Equation 1 and 2). The quantized values are used to obtain the transition probability matrix where each state is the quantized pair $(\hat{w}_i, \hat{d}_i)$. The matrix is structured in quadrants, highlighted by yellow and green in Figure 1.

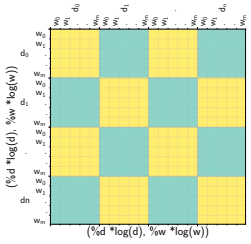


Figure 1: Transition Probability representation of a Model

Each quadrant represents a particular current delay $\hat{d}_i$ on the y-axis and a next delay $\hat{d_{i+1}}$ on the x-axis, these values are quantized in the range d_0 to d_n to keep the matrix from being prohibitively long. Each quadrant is further divided into smaller chunks representing the current values of $\hat{w}_i$ on the y-axis and a next window variable $\hat{w_{i+1}}$ on the x-axis, which are quantized in the range w_0 to w_n. Figure 1 shows an empty sample matrix. The transition probability for each chunk is computed by counting the number of occurrences of going from one state to another as $[(\hat{d}_i, \hat{w}_i), (\hat{d}_{i+1}, \hat{w}_{i+1})]$. We normalize each row within a quadrant so that all outgoing transition probabilities of any state would sum to 1.

2.5 MDI Implementation

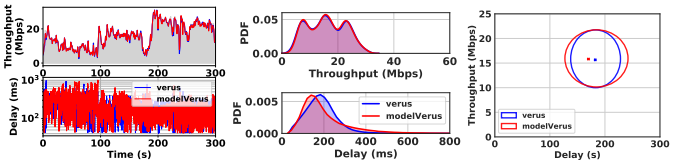
We implemented a generic sender and receiver in C that takes a transition matrix as an input and uses the matrix to decide the next sending window. The sender uses UDP as the transport protocol. It includes functions for calculating the packet delays based on the incoming ACKs and uses the delay to determine the sending window size after each epoch. Epoch time is when the algorithm updates the congestion window. Algorithm 1 outlines the MDI control loop. The model algorithm identifies the next sending window $\hat{w}_{i+1}$ in every epoch, obtained from the transition matrix, where a row within a quadrant of the matrix represents all possible values for the future sending window. MDI first identifies the operating quadrant through the row and column index. The row index is taken from the previous delay $\hat{d}_i$, and the column index from the current delay $\hat{d}_{i+1}$ (inferred from the incoming ACKs). Once the operating quadrant is identified, a particular row within the quadrant can be determined by the previous sending window $\hat{w}_i$. This row represents all possible sending windows decisions for the next epoch, each associated with a specific probability value. To decide the next sending window, MDI draws a random number (between 0 and 1) to determines the closest matching sending window. This process is a guided random-walk within the state transition probability matrix. If the values are outside the matrix dimensions, the next sending window is determined by a multiplicative increase/decrease to the current window to force it back to the matrix bounds (c_1 and c_2).

Algorithm 1 MDI pseudo-code

```
while TRUE do
    Compute d̂_{i+1} from ACKs
    if d̂_{i+1} < d̂_min then
        (Increase ŵ_{i+1} using const. multiplier c_1 > 1)
        ŵ_{i+1} ← ŵ_i * c_1
    else if d̂_{i+1} > d̂_max then
        (Decrease ŵ_{i+1} using const. multiplier c_2 < 1)
        ŵ_{i+1} ← ŵ_i * c_2
    else
        Determine matrix quadrant ← d̂_i and d̂_{i+1}
        Determine row within quadrant ← ŵ_i
        ŵ_{i+1} ← Randomly choose next state using transition probabilities in the chosen row
    sleep(epoch)                    ▷ epoch depends on the algorithm
```

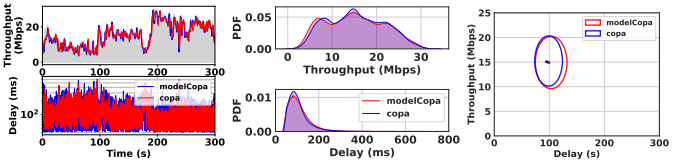
3 EVALUATION

We evaluated two CC protocols as a proof-of-concept of MDI: Verus, and Copa. These protocols are modeled through the training phase by generating the model transition matrix. The training is done using a set of 1000 different cellular traces (collected from real-world 3G/4G networks) that cover a wide range of network scenarios. To replay these traces, we used the MahiMahi [15] linkshell network emulator. We used a different set of cellular traces for testing, taken from several previously published papers:



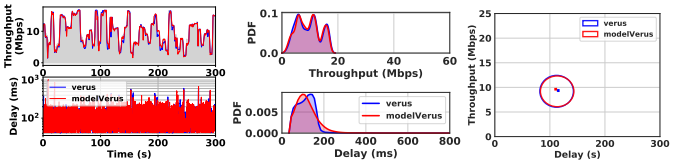
(a) instantaneous (b) PDF (c) Population

Figure 2: Verus Highway



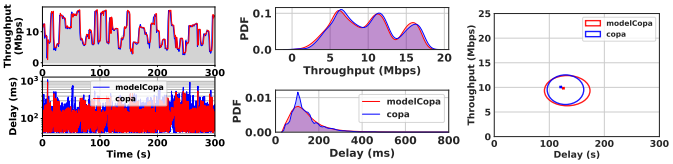
(a) instantaneous (b) PDF (c) Population

Figure 3: Copa Highway



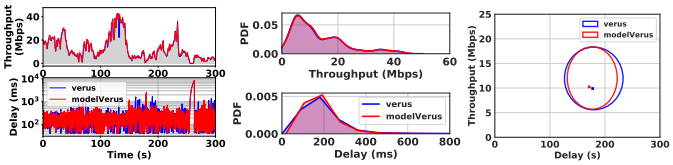
(a) instantaneous (b) PDF (c) Population

Figure 4: Verus Rapidly changing network



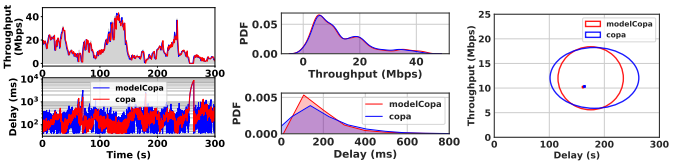
(a) instantaneous (b) PDF (c) Population

Figure 5: Copa Rapidly changing network



(a) instantaneous (b) PDF (c) Population

Figure 6: Verus 4G Verizon



(a) instantaneous (b) PDF (c) Population

Figure 7: Copa 4G Verizon

- 4G Verizon: taken from [27] and represents a recorded channel over Verizon's 4G network in the US.
- Highway: taken from [29], it represents a channel over a 3G network in the UAE while driving on a highway.
- Rapidly changing network: inspired by [5], this trace represents a network with a highly fluctuating channel, where the capacity varied randomly every 5 seconds.

We wanted to evaluate how well a model representation of an algorithm can track the throughput and delay of the native algorithms when run on the same network traces. This section shows the results for the MDI versions of Verus and Copa. For each protocol, we demonstrate the temporal variations of the original protocol against the MDI version of the protocol for a snippet of a 300 second run in one of the three scenarios in Figure 2a, 3a, 4a, 5a, 6a, and 7a. The results show that across both protocols (Verus and Copa), the MDI models (represented in red) are able to accurately track the throughput of the native protocol (represented in blue) temporally. In addition, the MDI models are able to temporally track the delay behavior of these protocols. To quantify that the MDI models statistically matches the characteristics of the original protocols, we computed the Probability Density Function (PDF) of the throughput and delay for both Verus and Copa respectively. Figure 2b, 3b, 4b, 5b, 6b, and 7b shows the PDF comparisons. It can be seen that the MDI throughput distributions match the native ones perfectly.

In summary, we observe that MDI have the ability to accurately track the throughput behavior of the two used protocols across highly variable network conditions, evident by the results of Figure 2c, 3c, 4c, 5c, 6c, and 7c. The figures show the overall summary comparing different values of the results population. Each of the MDI model and the native protocol is depicted by a circular shape representing the operational region of the protocol circumscribed by the 25% and 75% percentile of the obtained throughput and delay, where the crosses (x) indicate the median values. The lower and upper part of the shape represents 25% and 75% of the throughput, respectively (y-axis), whereas the left and right part of the shape represents the 25% and 75% of the delay, respectively (x-axis). Results show that MDI is capable of achieving quite similar statistical performance in terms of delay and throughput with a slight delay penalty not exceeding 5% (i.e., in the rapidly changing channel).

4 RELATED WORK

CC for cellular networks: conventional loss-based TCP variants, in particular Cubic [9], performs poorly in cellular networks. This is due to the high sending rate that fills up the buffers causing a bufferbloat [7]. Bufferbloat is detrimental to the performance of delay-sensitive applications like video calling. This has led to newer delay-based CC protocols like Sprout [27] and Verus [29] that were specifically designed in the context of cellular channels. While Sprout focuses on reducing self-inflicted queuing delays, Verus is designed to create a balance between the packet delays and the throughput. Recently, PCC Vivace [6], which followed PCC [5], has shown to react well to changing networks while alleviating the bufferbloat. PCC Vivace leverages ideas from online (convex) optimization in machine learning to do rate control. LEDBAT [21] is another delay-based CC algorithm developed for BitTorrent and other bulk-transfer applications that had limited adoption. BBR [3] was recently proposed by Google and has shown good results over cellular networks. BBR uses the bottleneck link's round trip propagation and bandwidth to find the optimum operating point for CC.

Applying machine learning to CC: new CC protocols being proposed have complex control loops, which makes them harder to understand in the context of different network conditions. The recent development of CC protocols that employ machine learning (e.g., Remy [26], Vivace [6] and Indigo [28]) have only compounded this issue (e.g., some of Remy's CC protocols employ rule tables with more than 100 rules). Weinstein et. al. (Remy) [26], Sivaraman et. al. [22] and Pötsch [19] have provided different methodologies to model non-linear CC from a theoretical perspective.

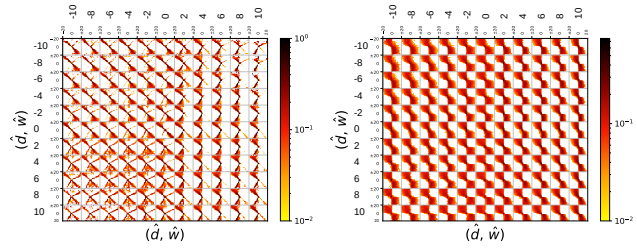
Analyzing TCP behavior: TCP and its variants have been thoroughly studied using the modeling and analytical techniques [4, 18, 20, 24, 25]. A recent work called ACT [23] uses the concept of a guided random walk in the state space of implementation variables, to find regions where the algorithm should never go, thereby indicating the existence of a possible bug in the implementation. Others also follow this approach of an automated model-guided method as well [12] to explore the variable space in the implementation of a CC algorithm. Our modeling approach also uses a random walk, but our state space is limited to a delay and window variable, and our goal is not to reach unreachable points but to guide the model to follow the native algorithm it is modeling.

5 DISCUSSION: WHY MDI?

5.1 Visualizing Protocols

The MDI transition matrix helps reason about the essence of the CC protocol behavior. Given that these matrices represent the probability distributions across the transition space, it highlights which states the protocol mostly operates in. It also shows how the protocol is likely to behave under specific network changes, such as an increase or decrease in the network delay. Verus and Copa's transition matrices (Figure 8a and 8b) clearly shows that the Verus matrix is less dense than Copa's, which means that Verus takes more decisive actions compared to Copa that tend to explore more. Each protocol shows a particular pattern that reflects the protocol's behavior; we call this the *protocol fingerprint*. The sectors in the matrix represent different transitions for a specific change in packet delay. The relative delay and window ranges are determined from the training phase, representing the 1% and 99% of the observed increase/decrease population.

The protocols' fingerprints reveal different characteristics of the protocol and how it reacts to various network changes. For example, the Verus transition matrix generally shows two distinct recurring patterns in the sectors: one in the left side of the matrix, and the other on the right side. We can see that the right side pattern mainly contains window decrease probabilities. This is consistent with Verus's design, where if the observed delay increases, Verus lowers the sending rate by moving the operation point down the delay profile curve.

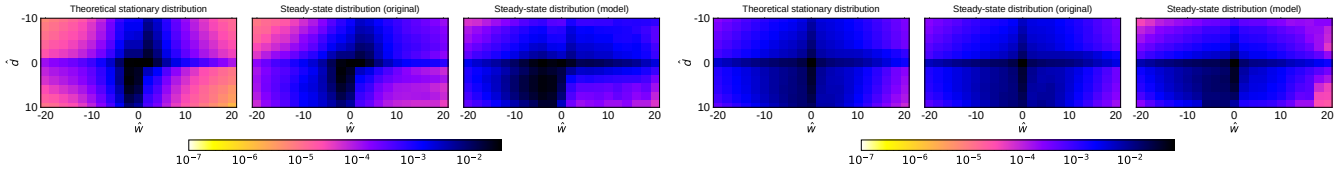


(a) Verus **(b) Copa**

Figure 8: MDI transition probability matrices

However, the left side pattern consists mainly of a diagonal line from the upper left corner down to the lower right corner. Additionally, the pattern also has an anti-diagonal, which becomes more dominant, moving down the sectors (i.e., when the delay feedback increases). This gives another insight to Verus. If a decrease in the previous delay is observed, it tends to continue alongside the same previous decision, extending the last window to decrease or increase. However, if Verus finds a delay-decrease with a prior increase in the delay, there is a higher probability that it might increase the window in the next decision despite the window decrease in the previous epoch. This confirms Verus's exploration behavior, where, in case of a delay reduction, it tends to increase the window to explore the channel variations immediately.

Copa's transition matrix, on the other hand, shows that the right side sectors of the matrix show almost the same pattern, with substantial probabilities in the upper left and lower right corners of the sectors and nearly no values in the top right or lower left edges. This means that regardless of the previous delay values or the severity of the observed delay values' increase, Copa tends to repeat its last epoch decision. For example, if Copa reduces the window, it will continue doing so in the next epochs. Unlike Verus, where it tends to minimize the window in case of an observed delay increase. Looking at the left sectors of Copa's matrix, we see that it has a similar pattern to the right side sectors with additional values in the upper right corner. These values become less dominant when moving down from the top to the bottom sectors. The sector's upper right corner represents increasing the window despite a reduction in the window in the previous epoch. Like Verus, Copa tends to increase the window by observing a delay reduction, and the severity of exploring increases, when the previous observed delays are decreasing.



(a) Verus **(b) Copa**

Figure 9: Comparison between the theoretical stationary (probability) distribution of the Markov chain model (left) that is trained on the training set of traces vs. the empirical distribution over the state space after mixing time for both the original and model versions of both protocols on the real world test traces. These are for mixing time threshold (ϵ) 10^{-3}.

Protocol	$\epsilon = 10^{-3}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-7}$
Verus	24	55	85
Copa	8	24	41

Table 1: Mixing times (in RTTs) for both protocols, calculated from the Markov model.

Testing protocol	$D_{KL}(P\|\|Q)$	$\max \|P - Q\|$
Copa	0.017	0.004
Model Copa	0.147	0.01
Verus	0.101	0.02
Model Verus	0.773	0.054

Table 2: KL Divergence of the steady-state distribution of the states (Q) in the testing set after mixing time w.r.t. the stationary distribution (P) computed from the Markov model, which is trained on the training set.

5.2 Convergence

Using our Markov formulation, we can provide convergence guarantees as strong as the original protocols, using properties of convergence of Markov chains. Before presenting our results, we briefly review some necessary notations and definitions regarding Markov chains and convergence.

Markov chains and Mixing times: Every Markov chain can be represented as a transition matrix P, where the entry p_{ij} represents the probability of transitioning to state j from state i. Suppose $\mu^{(t)}$ is row vector that represents a probability distribution over the state space at a time t. Then at $t+1$, the distribution over the state space is given by $\mu^{(t+1)} = \mu^{(t)}P$. If the initial distribution at $t = 0$ is given by $\mu^{(0)}$, then we have from above that $\mu^{(t)} = \mu^{(0)}P^t$. The limiting distribution λ is the limit of $\mu^{(t)}$ as $t \to \infty$. If a unique limiting distribution exists, then it equals the *stationary distribution*, which is the row vector π, such that $\pi P = \pi$. It is computed as the left eigenvector of the transition matrix corresponding to the largest eigenvalue [17]. The *mixing time* of a Markov chain, t_{mix}, is the time t to convergence from an initial distribution $\mu^{(0)}$, i.e., when the probability distribution $\mu^{(t)}$ over the state space is sufficiently "close" to the stationary distribution π that they are indistinguishable from one another. Any random walk process in a finite Markov space is associated with a finite mixing time [1]. To obtain the most conservative estimate, we define the mixing time as the maximum time to convergence starting from all possible initial states.

Observations: In our context, the state space comprises of the Cartesian product of 11 states in the delay space $\langle\hat{d}\rangle$ and 21 states in the window space $\langle\hat{w}\rangle$, a total of 231 $(\hat{d}, \hat{w})$ tuples. If the start state is i, then the initial distribution $\mu^{(0)}$ is a one-hot vector, with 1 at the location corresponding to state i and 0 everywhere else. Then, at every iteration t (equivalent to an RTT), we compute $\mu^{(t+1)} = \mu^{(t)}P$, and declare convergence at time t_{mix} when the maximum element-wise difference between $\mu^{(t_{\text{mix}})}$ and $\mu^{(t_{\text{mix}}+1)}$ is less than a certain defined threshold (ϵ). We compute mixing times for three different thresholds: 10^{-3}, 10^{-5} and 10^{-7}. The last is chosen as it approximately equals the machine epsilon for 32-bit float. Table 1 shows the mixing times (in RTTs) obtained from the transition matrix for both protocols.

The heatmaps in Figure 9 show the theoretical stationary distribution computed using the Markov chain transition matrix trained over a training sample of 1000 traces, compared with the empirical distribution of states *after convergence* (i.e. the mixing time) of the original protocols and the model versions over a separate testing sample of 60 cellular traces. The heatmaps are displayed over the two-dimensional $(\hat{d}, \hat{w})$ state space. The fact that these distributions match very closely is a robust result that our Markov model versions of the protocols are very close approximations of the original protocols. Table 2 shows the closeness of the two distributions in terms of the Kullback-Leibler Divergence [14] of the two distributions. The KL Divergence is a measure of how well one distribution approximates another. The closer the KL Divergence is to zero, the better the approximation. The table also additionally shows a simple maximum element-wise absolute difference between the two distributions. From the heatmap plots and these numbers, we observe that the model allows us to analyze the convergence properties of the original protocols, which has been known to be a challenging proposition for delay-based protocols due to complex non-linear control loops.

5.3 Drop-in replacement for CC protocols

The MDI representation essentially provides a *lookup table* of a protocol that can be plugged into the implementation of Algorithm 1. A model version of an algorithm can be used as a drop-in replacement for the actual algorithm and multiple such algorithms can be using the same reference implementation with different lookup tables. This

provides the ability to support dynamic congestion control switching that enables easy switching to a different CC protocols for different network conditions. If network conditions are known in advance, one can use model versions of a protocol known to perform well in such network conditions. In future work, we aim to test and extend the framework to capture the behavior of a broader array of congestion control protocols.

6 CONCLUSIONS

This paper describes the MDI framework that can approximate the behavior of delay based protocols and potentially aid in visualizing protocol behavior, understanding convergence properties and deriving a model-based protocol replacement. We hope that this Markov modeling approach provides a new lens for understanding the behavior of delay-based congestion control algorithms on highly variable networks. In future work, we hope to extend this framework to understand the behavior of a broader array of protocols, analyze fairness properties of MDI protocols and explore alternative state-space protocol representations within MDI.

REFERENCES

[1] David Aldous. 1983. Random walks on finite groups and rapidly mixing Markov chains. In *Séminaire de Probabilités XVII 1981/82*. Springer, 243–297.

[2] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical Delay-Based Congestion Control for the Internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 329–342. https://www.usenix.org/conference/nsdi18/presentation/arun

[3] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *Queue* 14, 5, Article 50 (Oct. 2016), 34 pages. https://doi.org/10.1145/3012426.3022184

[4] Neal Cardwell, Stefan Savage, and Thomas Anderson. 2000. Modeling TCP latency. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, Vol. 3. IEEE, 1742–1751.

[5] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA, USA, 395–408.

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

[7] Jim Gettys and Kathleen Nichols. 2011. Bufferbloat: Dark buffers in the Internet. *Queue* 9, 11 (2011), 40.

[8] Yihua Guo, Feng Qian, Qi Alfred Chen, Zhuoqing Morley Mao, and Subhabrata Sen. 2016. Understanding On-device Bufferbloat for Cellular Upload. In *Proceedings of the 2016 Internet Measurement Conference (IMC 16)*. Santa Monica, CA, USA, 303–317.

[9] S. Ha, I. Rhee, and L. Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008), 64–74.

[10] Zhenxian Hu, Yi-Chao Chen, Lili Qiu, Guangtao Xue, Hongzi Zhu, Nicholas Zhang, Cheng He, Lujia Pan, and Caifeng He. 2015. An In-depth Analysis of 3G Traffic and Performance. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges (AllThingsCellular '15)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/2785971.2785981

[11] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2013. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. ACM, New York, NY, USA, 363–374. https://doi.org/10.1145/2486001.2486006

[12] Samuel Jero, Endadul Hoque, David Choffnes, Alan Mislove, and Cristina Nita-Rotaru. 2018. Automated attack discovery in TCP congestion control using a model-guided approach. In *Proc. of Network and Distributed System Security Symp., San Diego, CA, USA*. 1–15.

[13] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. 2012. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of the 2012 Internet Measurement Conference (IMC '12)*. ACM, New York, NY, USA, 329–342. https://doi.org/10.1145/2398776.2398810

[14] S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *The Annals of Mathematical Statistics* 22, 1 (1951), 79–86. http://www.jstor.org/stable/2236703

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

[16] Ashkan Nikravesh, David R. Choffnes, Ethan Katz-Bassett, Z. Morley Mao, and Matt Welsh. 2014. Mobile Network Performance from User Devices: A Longitudinal, Multidimensional Analysis. In *Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362 (PAM 2014)*. Springer-Verlag New York, Inc., New York, NY, USA, 12–22. https://doi.org/10.1007/978-3-319-04918-2_2

[17] J. R. Norris. 1997. *Markov Chains*. Cambridge University Press. https://doi.org/10.1017/CBO9780511810633

[18] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. 1998. Modeling TCP throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review* 28, 4 (1998), 303–314.

[19] Thomas Pötsch. 2016. *Future Mobile Transport Protocols: Adaptive Congestion Control for Unpredictable Cellular Networks*. Springer.

[20] Charalampos (Babis) Samios and Mary K. Vernon. 2003. Modeling the Throughput of TCP Vegas. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '03)*. ACM, New York, NY, USA, 71–81. https://doi.org/10.1145/781027.781037

[21] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. 2012. Low Extra Delay Background Transport (LEDBAT). (December 2012). http://tools.ietf.org/rfc/rfc6817.txt RFC6817.

[22] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan. 2014. An Experimental Study of the Learnability of Congestion Control. In *Proceedings of the ACM SIGCOMM 2014 Conference*. Chicago, IL, USA.

[23] Wei Sun, Lisong Xu, Sebastian Elbaum, and Di Zhao. 2019. Model-Agnostic and Efficient Exploration of Numerical State Space of Real-World {TCP} Congestion Control Implementations. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 719–734.

[24] A. Wierman and T. Osogami. 2003. A unified framework for modeling TCP-Vegas, TCP-SACK, and TCP-Reno. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.* 269–278. https://doi.org/10.1109/MASCOT.2003.1240671

[25] Adam Wierman, Takayuki Osogami, and Jörgen Olsén. 2003. Modeling TCP-vegas Under on/off Traffic. *SIGMETRICS Perform. Eval. Rev.* 31, 2 (Sept. 2003), 6–8. https://doi.org/10.1145/959143.959146

[26] K. Winstein and H. Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference*. Hong Kong, China.

[27] Keith Winstein, Anirudh Sivaraman, Hari Balakrishnan, et al. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks.. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL, 459–471.

[28] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 731–743.

[29] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the ACM SIGCOMM 2015 Conference*. London, UK, 509–522.