Model-driven Delay-based Congestion Control for Cellular Networks

Thomas Pœtsch
NYU Abu Dhabi
Abu Dhabi, UAE
thomas.poetsch@nyu.edu

Yasir Zaki
NYU Abu Dhabi
Abu Dhabi, UAE
yasir.zaki@nyu.edu

Jay Chen
NYU Abu Dhabi
Abu Dhabi, UAE
jchen@cs.nyu.edu

Lakshminarayanan Subramanian
New York, USA
lakshmi@cs.nyu.edu

Delay-based congestion control protocols appear to respond more efficiently to the unique properties of cellular networks compared to loss-based protocols. Unfortunately, delay-based congestion control protocols are often hard to understand and interpret by protocol designers due to their non-linear nature. As a result, existing models tend to include many oversimplifications and assumptions that limit their applicability to real cellular networks.

In this paper, we develop a new stochastic two-dimensional discrete-time Markov modeling approach that dramatically simplifies the understanding of delay-based congestion control protocols. This model allows us to analyze a protocol's behavior and to replicate its analysis under different network scenarios. We use the Verus congestion control protocol as a case study to demonstrate that the model's performance matches that of Verus. The modeling approach developed in this paper is extensible to a variety of different settings and protocols. We describe the central elements that a designer should specify to achieve comparable performance with other delay-based congestion control protocols.

Keywords

delay-based congestion control, analytical modeling

1. INTRODUCTION

Cellular networks present unique challenges to congestion control algorithms not found in wire-line or datacenter networks. Unpredictable losses at relatively short timescales and other fluctuations cause loss-based algorithms to overreact and underperform. In contrast, delay-based congestion control protocols appear to respond more efficiently to the properties found in cellular environments.

Recently, several delay-based congestion control protocols have been proposed in the context of cellular networks, such as Sprout [18] and Verus [19]. These algorithms substantially outperform other congestion control schemes. Though these protocols were extensively evaluated, there is still a lack of deeper insight as to why they work and their non-linearity makes them challenging to model analytically.

Inspired by Remy [17], this paper attempts to bridge the gap between delay-based congestion control protocol performance and our understanding. We introduce a new stochastic two-dimensional discrete-time Markov model that simplifies the complexity of delay-based congestion control protocol design. Our model is implemented in the OPNET simulator and can serve as a drop-in replacement for delay-based congestion control protocols.

To evaluate our model, we chose the Verus protocol as a comparative case study. We trained our model using channel traces from real cellular networks and show that it can be used in networks with similar conditions to achieve comparable performance to Verus in terms of throughput and packet delay.

As in Remy, we find that the 'design range' and knowledge of the network is a critical element of training. In our model, the dominant factor in achieving good performance is matching the number of users in the training and evaluation network configurations; matching user models achieves similar performance, but non-matching user models results in increased network delay. Since it is not always possible to have a priori knowledge of the network in which the model will to operate, we present an approach that combines the training sets with multiple users. Using this approach, our model's performance is restored and comparable to state-of-the-art congestion control protocols like Verus. The conceptual framework and our model may be similarly applied to other delay-based congestion control protocols.

2. BACKGROUND AND RELATED WORK

One of the most widely used congestion control protocols is the Transmission Control Protocol (TCP). Over the years, numerous other TCP versions have been proposed, including: TCP Cubic [9], TCP Westwood [5], and TCP Compound [15]. In total, over thirty different TCP congestion control variants exist today. All of these congestion control protocols attempt to adapt TCP's basic guarantee of reliable end-to-end data delivery to the variety of Internet environments.

A simple taxonomy of congestion control protocols consists of: loss-based, delay-based, and hybrid congestion control protocols. Delay-based congestion control protocols, such as TCP Vegas [3], have recently gained increased attention in the context of cellular networks.

2.1 Related Work

Although TCP has remained a gold standard for many years, there has been a consistent stream of new transport and congestion control protocols with various objectives and design goals.

LEDBAT [13] is a delay-based congestion control algorithm with the goal of minimizing the packet delay and fully utilizing the available bandwidth of bulk-transfer applications. It was specifically designed for the BitTorrent network but is now used by different companies across the world.

PCP [2] is an efficient congestion control protocol that aims to minimizing packet delay and packet loss while providing high stability under extreme load and fairness among competing flows. PCP makes use of sending explicit probes into the network to explore the available bandwidth. It starts with a specific sending rate and doubles the rate for every successful probe. Sprout [18] is a recent end-to-end transport protocol that was specifically designed for the context of cellular networks for streaming applications that demand high throughput and low packet delays. Sprout focuses on the problem of reducing self-inflicted queuing delays and uses packet inter-arrival times to detect congestion. It then makes short-term forecasts of upcoming changes in the link rate by using probabilistic inference. Sprout achieves a significant reductions in the end-to-end delay in cellular networks while maintaining good throughput characteristics.

PCC [6] is a novel congestion control protocol that achieves large performance improvements, better fairness and stability compared to legacy TCP. The protocol aims at observing continuously the relationship between its sending behavior and the experienced performance on rapidly changing networks. QUIC [8] is a secure transport protocol developed by Google and implemented in the Chrome browser that runs on top of the UDP and integrates ideas from TCP and TLS/DTLS. Since it is based on UDP, QUIC implements its own congestion control algorithm and several features from the application protocol SPDY [7]. The focus of QUIC is to reduce connection and packet delays and hence improve the performance of connection-oriented web applications that are currently using TCP. Thus, QUIC can more be viewed as an application protocol that is optimized for HTTP traffic.

Remy [17] is a computer-generated congestion control algorithm that generates congestion control algorithms offline. The goal of Remy is to find the best congestion control algorithm that solves the problem of when an endpoint should transmit a packet. It uses a novel approach to model end-to-end congestion control in a multi-user network. A machine can be trained offline to learn congestion control schemes that suits a specific 'design range' of operation. Remy has been demonstrated to outperform human designed congestion control algorithms. However, the performance of the designed algorithm is determined by the design range and the knowledge of the network and the traffic characteristics of the network.

Under the umbrella of Remy, Sivaraman et. al. [14] investigated the learnability of congestion control under imperfect knowledge of the network. In this work, a experimental study was conducted using the automated protocol-design of Remy. The authors found that in model-driven congestion control, simplifications of the network characteristics are acceptable and that minor deviation of the design range and the network of operation still yields good performance. However, the authors highlighted that a match of the number of senders in the training of the protocol and in the network of operation is essential to obtain reasonable performance. In our work we make similar observations and also propose a solution to this issue.

2.2 Challenges modeling Delay-based Congestion Control Protocols

Existing analytical models in the literature made several attempts to describe delay-based congestion control protocols. Samios et. al. [12] model the throughput of TCP Vegas under the assumption of a stable baseRTT throughout the connection. In [16], the authors present a general analytical framework to model and analyze several TCP variants, including TCP Vegas. The model assumes a TCP flow under on-off traffic and bulk transfers. The author of [10] proposes a model that estimates the operating point of TCP Vegas sources under on-off traffic.

In summary, the available analytical models for variants of TCP describe only the basic functionality of delay-based congestion control protocols and assume simple and relatively static network models. Due to the design goals of Verus [19] to operate on networks with highly fluctuating channel conditions, the available models are inappropriate for protocols like Verus. Fur-

thermore,there are no network models available that accurately reflect the properties of cellular networks. This leads to simpler models that do not realistically reflect the properties of cellular networks. Furthermore, because Verus bases its decisions on a delay profile curve that changes over time an analytical model of the Verus protocol can be very complex.

3. MODEL-DRIVEN CONGESTION CONTROL

At an abstract level, any delay-based congestion control protocol can be modeled as a relationship between the network delay and the sending window size. Let us assume that for a delay-based congestion control protocol in a particular network configuration, there exists a region of value pairs in a two-dimensional space. Each value pair consists of the network delay d and the sending window size w. The size of the region is determined by the channel conditions of the network and hence is upper bounded by the maximum delay d_{max} and the maximum sending window w_{max}. Such a region has a square shape and all value pairs that the delay-based protocol has used or will use reside within this region. Figure 1 shows a schematic diagram of an example of possible regions where the green regions represent fixed networks and the red regions represent cellular networks.

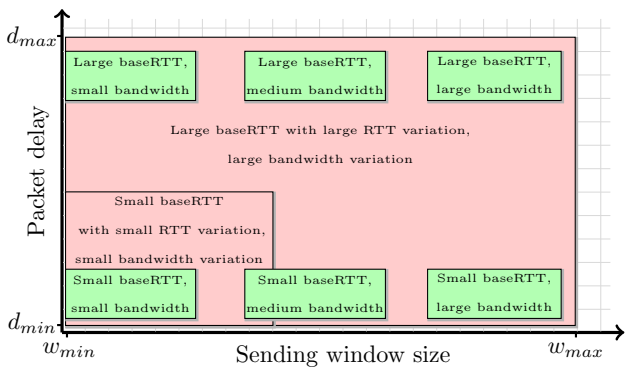


Figure 1: Schematic diagram of possible value pair regions for packet delay and sending window size

The idea behind this model is to describe any arbitrary delay-based congestion protocol by states of a discrete-time Markov chain. The state space of the Markov chain is given by the region of the underlying network and is generated offline with a specific training configurations. Using this state space, the protocol can be emulated through a guided random walk within this state space.

The outcome of the model represents a description of a delay-based congestion control protocol that obeys similar properties. That is, it simplifies the protocol description significantly and reduces the implementation complexity.

Each state of the Markov model represents a value pair of network delay and sending window size. These value pairs form a monotonically increasing curve within the network region. The Verus protocols refers to this curve as the *delay profile* and describes the properties of the network at every time instance precisely. In general, the delay profile relationship is a monotonically increasing – but not necessarily a linear – function. However, the way each protocol estimates the sending window size may be different. TCP Vegas, for example, observes the instantaneous packet delay and adjusts its sending window based on its congestion avoidance algorithm at any time when packets are being sent. Verus, on the other hand, maintains a delay profile curve based on the recent history of the network conditions. This delay profile curve varies over time and is updated by the instantaneous network conditions.

3.1 Discrete-time Markov Model

As stated earlier, a delay-based congestion control protocol is modeled by a discrete-time Markov model with states representing a value pair of the network delay and the sending window size. The set of the states ($\mathcal{M}$) is defined by a region as:

$$\mathcal{M} = \{(d_{min}, w_{min}), ..., (d_{max}, w_{max})\} \quad (1)$$

However, as the region highly depends on the channel conditions of the network and the behavior of the congestion control protocol, not all value pairs of the region are necessarily used throughout the connection.

The states of the two-dimensional Markov chain are described in the form of tuples with value pairs of d and w. Thus, a state is defined as:

$$S_{kl} = \{(d_k, w_l), (k, l) \in \mathcal{M}\} \quad (2)$$

where $\mathcal{M}$ is the set of all states and k and l are the indices of d and w, respectively.

Likewise, another state is defined as:

$$S_{rv} = \{(d_r, w_v), (r, v) \in \mathcal{M}\} \quad (3)$$

where r and v are the indices of d and w of the state.

Due to the memoryless property of Markov chains, a transition from the current state to the next state only depends on the current state. Here, this can be mapped to S_{kl} being the current state and S_{rv} being the next state. Hence, a transition can be written as a transition probability of:

$$p_{(k,l),(r,v)} = p[S_{rv}|S_{kl}]. \quad (4)$$

By looking at all possible state transitions throughout the entire connection, a two-dimensional discrete-time

Markov chain can be generated to represent the state transitions. All values within each row (S_{kl}) sum to one since the row covers all possible transitions leaving the state S_{kl}. Hence, for all rows, the following equation holds:

$$\sum_{r=d_{min}}^{d_{max}} \sum_{v=w_{min}}^{w_{max}} p_{(k,l),(r,v)} = 1. \quad (5)$$

The resulting two-dimensional state transition matrix $\mathbf{P}$ describes precisely the transition probabilities of the protocol from any state S_{kl} to any other state S_{rv}, including self transitions.

Since each state represents a value pair of d and w and transitions among all combinations (within the above specified boundaries) of d and w are possible, the size of the state transition matrix has the dimension $m \times m$, where

$$m = (d_{max} - d_{min} + 1) \cdot (w_{max} - w_{min} + 1). \quad (6)$$

An example of a state transition matrix with arbitrary probabilities is shown in Figure 2. It can be seen that all combinations of d and w are represented as a state and hence the rows and columns are separated into subsets (only two are shown in the figure). Each subset represents one particular d with all possible w's, ranging from w_{min} to w_{max}. However, transitions from one subset to any other subsets are possible (and required).

Next state S_{rv}

Present state S_{kl} \ d_v, w_n	d_{min}, w_{min}	$d_{min}, w_{min}+1$	...	d_{min}, w_{max}	$d_{min}+1, w_{min}$	$d_{min}+1, w_{min}+1$	...	$d_{min}+1, w_{max}$	...	d_{max}, w_{max}
d_{min}, w_{min}	0.01	0.02	...	0.72	0.02	0.11	...	0.06	...	0.01
$d_{min}, w_{min}+1$	0.31	0.54	...	0.07	0.01	0.02	...	0.04	...	0.03
...	...	...	...	...	...	...	...	...	...	...
d_{min}, w_{max}	0.2	0.1	...	0.09	0.03	0.09	...	0.04	...	0.01
$d_{min}+1, w_{min}$	0.4	0.08	...	0.03	0.09	0.41	...	0.01	...	0.31
$d_{min}+1, w_{min}+1$	0.72	0.01	...	0.06	0.04	0.05	...	0.11	...	0.05
...	...	...	...	...	...	...	...	...	...	...
$d_{min}+1, w_{max}$	0.61	0.07	...	0.09	0.04	0.01	...	0.02	...	0.01
...	...	...	...	...	...	...	...	...	...	...
d_{max}, w_{max}	0.23	0.01	...	0.05	0.29	0.33	...	0.09	...	0.02

Figure 2: Example of a transition probability matrix with arbitrary transition probabilities

The obtained Markov chain serves as an input for the model implementation. Further, the model makes use of the state transition matrix to determine, in regular time intervals (i.e, epochs), how many packets to send. Hereby, the model applies a guided random walk in the state space of the state transition matrix and takes delay feedback in term of acknowledgments from the receiver into account (explained in the next subsection).

3.2 Guided Random Walk

A random walk is a mathematical description of random steps to move from one state to another state within a Markov chain. In a guided random walk, the state transition matrix is divided into subsets (as described above) and the transition from the current state S_{kl} to the next state S_{rv} is determined by a subset of the matrix that represents states within a desired range. That is, the next state S_{rv} resides within this subset and is determined by a minimum distance of the transition probabilities within the subset to a random number. Given this, a pre-selection of the range in which the next state S_{rv} resides is assumed and hence called a "guided" random walk.

More precisely, the guided random walk describes stochastically a change of the sending window from w_l to w_v under the condition that d_k changes to d_r. While the transition of d is determined by the change of the observed packet delay, the transition of w is based on a random process.

As stated earlier, the model operates in regular intervals, so-called epochs in which it decides how many packets to send (or abstain). Hence, at the beginning of every epoch, the protocol resides in one particular state S_{kl} (row of the state transition matrix). This state S_{kl} is determined by the packet delay d_k and the sending window size w_l of the last epoch.

In the current epoch, the most recent maximum perceived packet delay d_k (observed during the last epoch) is used to determine the subset of the next state S_{rv}. This is accomplished by defining a subset ($\mathbf{P}_{subset}$) of all states S_{rv} that correspond to the currently perceived packet delay d_r and all possible values of $w_{min} \leq w \leq w_{max}$. Again, the transition from state S_{kl} to state S_{rv}, and consequently from d_k to d_r, is defined by the change of the packet delay during the last epoch. Further, since the subset is defined by the most recent delay d_r, all values in the subset reflect possible transitions for the delay change. According to Figure 2, a subset corresponds to one specific colored region in the figure (red or blue) and has always the length of $w_{max} - w_{min} + 1$.

In order to obtain the next state S_{rv} and hence the next sending window size w_v, a random number N from a uniform distribution is drawn. The interval of N is $[min(\mathbf{P}_{subset}),\ max(\mathbf{P}_{subset})]$ and represents all probabilities within $\mathbf{P}_{subset}$. The state S_{rv} is assigned by finding the minimum distance of all states in $\mathbf{P}_{subset}$ to N and hence the state with the closest match of the random number N. As the states within $\mathbf{P}_{subset}$ reflect all w's for the recent delay d_r, the state with the closest match determines w_v.

In the next epoch, this procedure is repeated by taking the value pair d_r and w_v as d_k and w_l, respectively, and by assigning a new $\mathbf{P}_{subset}$ from the currently observed packet delay. Finally, a new random number is drawn to obtain the the next w_v.

3.3 Model Framework

The model-driven congestion control concept is shown in Figure 3. As seen in the figure, the model takes several input parameters into account in order to build the discrete-time Markov state transition probability matrix. The input of the model incorporates parameters that describe the relationship between the observed network delay and the size of the sending window. It utilize a similar concept to Remy's [17] training phase, where a certain "design range" of operation is assumed and values are obtained from samples of a large set of simulations with predefined network and client configurations. This is denoted as the *training* phase. Unlike other mathematical models, the training set in this paper also reflects networks with fluctuating channel conditions (channel traces from real cellular networks). The exact details on how the training is handled is explained in the next section.

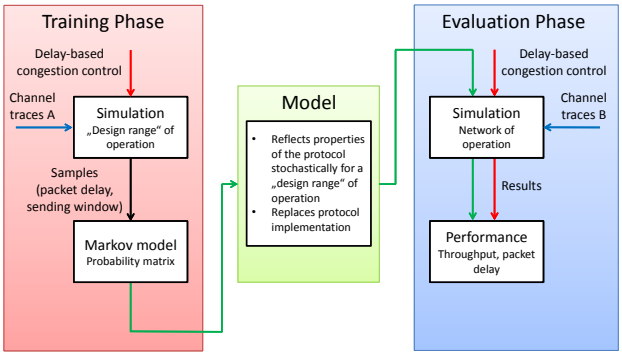


Figure 3: The concept of modeling delay-based congestion control protocols

The output of the training phase is a discrete-time Markov chain that describes the transition probability properties of all samples from the network delay and sending window size of the training phase.

To compare the performance of the model, its implementation is run on endpoints in environments with a different set of network and client configurations. This "network of operation" is denoted as the *evaluation* phase and should exhibit similar network and client configurations as the configurations in the *training* phase.

4. MODEL TRAINING

The model is trained in two steps: The first step is to collect the samples from a simulation or real-world deployment running the delay-based congestion control protocol. For simplicity and to remain generalizable, the samples were collected from a large set of simulations. Several hundred thousand samples were collected and processed in the second step to obtain the state transition matrix.

4.1 Samples collection

The samples are generated from simulations using different scenarios, each configured with a specific number of Verus users and a different network configuration (represented by channel traces). These channel traces represent the "design range" of operation and are also referred to as the *training* channel traces.

In order to construct the state transition matrix, the samples were collected and post-processed as follows: For every packet that Verus sends, the value pairs of the current packet delay d (with a granularity of 1 ms) and the number of sent packets w were recorded during the simulations.

Next, all transitions from one value pair to the next value pair inside the recorded dataset were counted. A value pair of d and w is considered as a state and hence the resulting matrix reflects the number of transitions from one particular state to any other state. Hereby, a row of the matrix corresponds to the current state S_{kl} and a column to the next state S_{rv}. Consequently, the corresponding probabilities of the state transition matrix can be calculated by normalizing all elements per row and applying Equation 5.

4.2 State Transition Matrix

Due to the structure of the matrix where for every value of d, all available values of w are present, the dimension of the matrix with a granularity of 1 ms becomes extraordinarily large (compare Equation 6 and see Table 2). Besides that, when constructing a state transition matrix from a set of samples (no matter where they are from), empty transitions are often observed and hence result in a very sparse matrix. Consequently, the matrix contains a significant amount of empty transitions that increase the resulting file size of the matrix enormously. This circumstance can be optimized by reducing the value ranges of d and w accordingly and was accomplished by the following two techniques:

- **Truncating**: Spurious outliers of high packet delays in the samples are removed in order to reduce the value range of d. In this model, the 99.9% percentile of the d values were used.
- **Quantizing**: Despite truncating of the values of d, the range of values is still very large due to the granularity of 1 ms. Hence, the d values are quantized by 2, 5, 10, and 20 ms.

The outliers of the samples that are removed by truncation, originated from very specific situations where peaks in the packet delay were observed. These outliers were, however, very rare and hence did not represent significant importance of the statistical transition properties of the samples. On the contrary, quantization introduces a slight penalty on the statistical properties of the samples. Nonetheless, with both techniques described above, the resulting size of the matrix could be

significantly reduced but also adds a marginal degradation to the performance of the protocol.

5. MODEL IMPLEMENTATION

Our model is implemented in OPNET [11] and contains a sender and a receiver application. The sender consists only of a few lines of code and uses UDP as the underlying transport protocol. The application calculates the packet delay based on incoming acknowledgments, applies the guided random walk based on the state transition matrix, and exhibits a timeout timer. The application is intentionally kept simple because the core behavior of the protocol is defined by the state transition matrix, which does not change over time. Furthermore, the sender needs only to maintain a sending window and to transmit the corresponding packets because the number of packets to send is solely determined by the state transition matrix. The sending of packets (and also the determination of the sending window) occurs at fixed time intervals (epochs) by applying the guided random walk in the state transition matrix and is based on the delay feedback from the receiver. Further, no slow-start is required as the delay feedback of the first packet is immediately applied to the guided random walk.

The receiver is also straightforward and only sends acknowledgments back to the sender. Packet or reordering support is not yet implemented.

5.1 Applying the Guided Random Walk

At every epoch, the guided random walk is applied in order to find the next value for w. Hereby, the last state in the state transition matrix (row) that corresponds to the packet delay and the number of packets that were sent during the previous epoch is used as a starting point (origin). Based on the recent packet delay (measured during the last epoch), the subset of the next states is defined. The subset of this particular packet delay contains all possible values of w and, hence, all transitions from the origin to one of these w values. Next, a random number from a uniform random number generator is drawn to find the w with the closest match in terms of transition probability. The obtained w is then sent to the network and its value, together with the current delay, is stored as the current state. This state will then be used during the next epoch as the previous state and the procedure starts from the beginning with a subset of the packet delay that was experienced during the current epoch.

In cases where the desired subset does not contain any values, i.e., no transition from the current state to any of the w in the subset were present in the samples, there are only two options without violating the random walk: either going in the matrix to the left or right by one interval (i.e., deceasing or increasing the current delay), or going one (or more) rows up or down (i.e., changing the w of the origin). While the second option has only minor impact on the performance, the first option could severely change the performance, especially when going to the left, i.e., assuming a smaller current delay. Based on observations from the generated state transition matrices, the matrix is more sparse in the higher delay regions and hence it is very unlikely to find a data point in neighboring intervals. Hence, the second option is applied according to the following procedure: First, the w value of the state from the previous epoch is decremented by one, which means moving one row up in the matrix. Although this w does not correspond to the real number of packets that were sent during the last epoch, the probability of finding a data point in the desired subset increases. As long as the decrement of w only occurs within the same delay, this has only a minor influence on performance and underestimates the previous throughput of the last epoch. Second, when finding a subset with more than one value, a random number is drawn and the w with the closest match in terms of transition probabilities is chosen (and sent).

6. EVALUATION

In this section, the performance of the model implementation is evaluated and compared to the performance of the modeled delay-based congestion control protocol Verus. Both implementations were run independently in the simulator. The evaluations highlight the results of throughput and network delay of both evaluations and cover five different channel traces that were obtained from commercial 3G cellular networks. The channel traces that are used for the evaluation are different from the ones used to train the model.

Further, the model is investigated in scenarios with a non-matching user model. That is, the model was trained with a certain number of users and evaluated in a scenario with a different number of users. Hereby, all users execute the same non-matching user model that is not particularly trained for the number of users in the evaluation scenario. Likewise, the performance of the model is also investigated in scenarios with a matching user model, i.e., the model was trained with the same number of users as in the evaluation scenarios. Finally, the performance of the model is also evaluated with a combined user model of several users and evaluated in scenarios with different number of users.

6.1 Evaluation Setup and Methodology

The basic setup of this evaluation is depicted in Figure 4 and is implemented in the simulation environment OPNET [11]. The network consists of one server and one or more clients, where both are running the corresponding implementation. In order to compare the performance of throughput (Mbit/s) and network delay

(ms) of the model and the Verus protocol, the same simulation was executed independently with the model and with the Verus protocol. The network is kept very simple and the topology consists of two or more endpoints that are connected via two switches. While one switch is a regular switch that is not congested, the second switch is used to shape the traffic according to the collected channel traces. Further, the second switch implements a Random Early Discard (RED) Active Queue Management (AQM) with a packet drop probability of 0.1% and a minimum and maximum average queue length of 3 Mbit and 9 Mbit, respectively. These parameters were chosen based on suggestions in [4]. The training of the model assumes a certain set of parameters to create and to evaluate the model. Table 1 summarizes all model assumptions and simulation parameters. Using the OPNET simulator, the channel traces were fed into a traffic shaper[1] and were replayed upon packet arrival. The parameters of the Verus protocol were set to the default values.

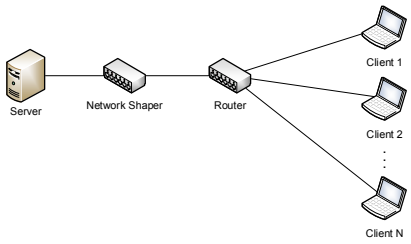


Figure 4: Simulation setup with one server and one (or more) clients

The reason why Verus is used in this evaluation as a comparison is twofold: First, Verus has been designed for networks with highly fluctuating channels and hence is most suitable for the given channel traces. Second, the OPNET simulator does not include any other delay-based congestion control protocol. Additionally, the idea of the model is to build a mathematical description that reflects the properties of delay-based congestion control protocols in a statistical fashion. Hence, a direct comparison to Verus is helpful to see whether the statistical model is valid and if comparable performance can be achieved.

Each simulation of the model was run for 300 s and was repeated five times with different random seeds. By means of that, the mean values, standard deviations and the 95% confidence intervals per flow are calculated. Since the resulting confidence intervals for the throughput and network delay were rather small in most scenarios, they are not displayed in the graphs of this evaluation. Furthermore, since Verus does not contain any

[1]The traffic shaper is a modified version of a network router in OPNET.

Parameter	Value(s)
Generator protocol	Verus
Verus parameters	Default
Training phase	
Network model	Channel traces set 1
Number of channel traces	5
Traffic model	Full buffer
Active Queue Management (AQM)	Min queue length: 3 Mbit Max queue length: 9 Mbit Drop probability: 0.1%
Simulation time per trace	300 seconds
Number of users per model	1, 2, 3, 4
Collected samples	Network delay d and portion of sending window w
Number of collected samples per user model	1: ∼290 000 2: ∼580 000 3: ∼850 000 4: ∼920 000
Truncation of the matrix	99.9%
Quantization of the matrix	2, 5, 10, 20
Evaluation phase	
Network model	Channel traces set 2
Number of channel traces	5
Traffic model	Full buffer and ON/OFF model
Simulation time per trace	300 seconds
Number of users	1, 2, 3, 4
UDP packet size	1450 bytes
Confidence interval	95%

Table 1: Model assumptions of the training and the evaluation phase

random number generator in its implementation and due to the fact that the channel model is exactly modeled by the given channel traces, the results of Verus are only run once and no confidence intervals are therefore given.

As stated in Section 4, the training of the model was done with a certain set of channel traces that emulate real cellular networks. The length of the training was 300 s per channel trace and results in a total training time of 1500 s. Due to the variety of the different channel traces, this time reflects a wide range of channel conditions. In the evaluation, a different set of channel traces is used that was collected under similar conditions but at different times. However, the throughput and the network delay that a protocol experiences on these channel traces differs from the channel traces used in the training.

Further, although the state transition matrices were generated by the Verus protocol, it is a valid approach to compare the results against Verus since the network properties are different.

6.2 Effect of Quantization

As explained in Section 4.2 and listed in Table 1, the number of collected samples per number of users per scenario is in the order of several hundred thousand. Even after quantization and truncation, the resulting

state transition matrix became considerably large. This section investigates the properties of the resulting state transition matrices and the impact of quantization on the performance of the model.

Table 2 summarizes the matrix dimensions and the resulting file size of the comma-separated values file for quantizations of 1, 2, 5, 10, and 20 ms.

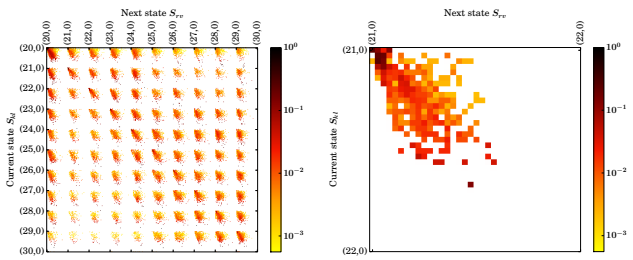
Quantization	1 ms	2 ms	5 ms	10 ms	20 ms
Matrix size (m)	25488	12744	5112	2556	1296
File Size	2.4 GB	620 MB	100 MB	25 MB	7 MB

Table 2: Matrix and file sizes of different quantizations for a training with one user ($\sim$ 290 000 samples)

Figure 5a shows the state transition matrix for the training scenario with one user and with a quantization of 1 ms. The range of the samples varies from 20 to 727 ms for d and from 0 to 35 packets for w. According to Table 2, this results in a matrix dimension of 25488×25488 elements. Due to this, the figure is limited to the first ten quantization steps (i.e., 360×360 elements). Consequently, the first 9 ms of the samples ($20\ ms \leq d \leq 29\ ms$) are shown. Each interval in the figure represents 36 values of w ($0 \leq s \leq 35$). The data points in the figure represent the probabilities of the transitions from a current state S_{kl} to the next state S_{rv} and are depicted in logarithmic scale[2]. That means, the darker the color, the higher the transition probability. A white data point represents no transition from the current state S_{kl} to the next state S_{rv}. This is, however, highly dependent on the scenario in which the samples were collected and might change for another set of scenarios. In addition, Figure 5a shows that most of the transitions appear almost equally in all visible states. However, it can be seen that some regions (intervals) are more sparse than others, especially in the lower left corner. That is due to the fact that transitions from a larger d value to a smaller d value do not occur that often. These transitions only occur when the channel conditions change from good to bad within very small time scales, i.e., within one epoch of 5 ms. Since Verus is slightly more conservative in decreasing the sending window (δ_1 and δ_2 parameter), this transition happens in smaller steps than transitions to higher delay values.

When zooming into one particular interval, e.g., 21 to 22 ms, it can be seen from Figure 5b that a significant amount of transitions are self-transitions and transitions to very neighboring states. In particular with smaller w values, a higher probability to remain in an equal or similar state is observed. Further, it can be seen that a considerable amount of transitions remain

[2]Due to the small values of the probabilities (the sum of each row equals one), a logarithmic scale allows for a better separation of the individual values.



(a) Full scale (b) Zoomed version

Figure 5: State transition matrix of transitions for the samples of one user and a quantization of 1 ms.

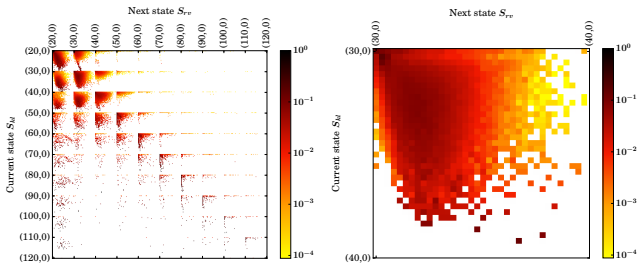
empty in this particular interval; and also in many other intervals.

Now to highlight the effect of quantization, Figure 6a shows an example of the state transition matrix for the same set of samples with a quantization of 10 ms. Again, due to the resulting size of the matrix after the quantization (i.e., 2556×2556 elements), Figure 6a is limited to the first ten quantization steps (360×360 elements). The figure shows that the smaller d (and also w) is, the more data points are present and the higher the probabilities are. Smaller d values are located in the upper left corner of each quantization step (interval). Furthermore, it can be observed that a diagonal colored region is formed from the upper left to the lower right corner of the figure. This region occurs due to the quantization of the delay, i.e., combining ten values to one. That is, the figure collates (or compresses) the visible states of Figure 5a into one region in the upper left corner. As a result it can be observed that the upper left corner shows a larger amount of transitions than other regions of the figure. Regardless of w, this effect can be explained by the fact that the probability is significantly higher to transit from a state with a particular delay to another state with the same or similar delay; including self-transitions. That means, in the majority of the transitions, the delay remains constant (when neglecting the quantization effect) and a transitions happens among the w values only. However, due to the quantization, a small error is introduced but also increases the density of the matrix in the region with smaller (and more often occurring) network delays. This region, however, can shift depending on the training of the model.

Figure 6b depicts a zoomed in version for the interval from 30 ms to 40 ms. Due to the quantization, the region in this interval is more dense when compared to Figure 5b. Also, a higher probability to remain in an equal or similar state is observed for smaller w values.

6.3 Training with a Non-matching User Model

The aim of this section is to show the influence of a non-matching training of the model on the protocol



(a) Full scale (b) Zoomed version

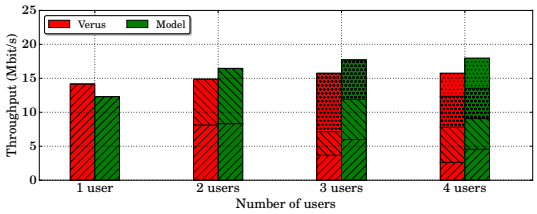
Figure 6: State transition matrix of transitions for the samples of one user and a quantization of 10 ms

performance. That is, the model was trained with a different number of users as during the evaluation. More precisely, the samples of the model were generated from simulations with one user. The resulting state transition matrix is used in this section as an input for the model.

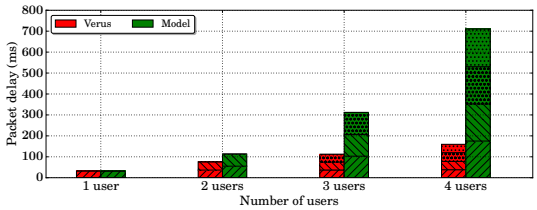
The evaluation of the model is investigated in scenarios with different number of users, namely two, three, and four users. That means, in each scenario, either two, three, or four users are present in the network and execute the model implementation with a matrix that was trained with one user. As in the previous section, the quantization for this evaluation was set to 2 ms and all other parameters were set as stated in Table 1. In order to limit the amount of results in this section, only the results of channel trace #1 are shown. This channel trace serves as an example to highlight the effect of a non-matching user model. The results of the other channel traces show similar or slightly lower network delays and similar or slightly higher values for the throughput.

Figure 7 shows the mean values of throughput and the packet delay of the model with a training set of one user. The protocol was run in scenarios with a non-matching number of users. In the scenario with one user, both Verus and the model show an almost equal network delay. Verus, however, shows a slightly higher throughput since Verus actively explores the available channel capacity constantly and reacts faster to the channel changes.

Now, when increasing the number of users in the network to two, three, and four it can be seen that the model achieves slightly higher throughput but also experiences an increase in the network delay throughout all three scenarios. As the model was trained with one user, the model assumes in all scenarios that only a single user is present and hence uses a state transition matrix with mismatching probabilities. While this effect is almost negligible in the scenario with two users, the effect becomes more visible when the number of users is increased. Especially in the scenario with four users, the packet delay of each individual user in the network is increased by a factor of four. This effect is mainly caused by the fact that in the training scenario with one user, a smaller delay causes the protocol to send more packets than it is supposed to do in a scenario where more users are present. As a result, states with small delays in the state transition matrix imply more (and larger) probabilities for higher w values. Consequently, the mismatching user model causes a sending of packets at the wrong times (epochs) and therefore results in higher delays. However, the model achieves almost perfect fairness among users despite a non-matching user model.



(a) Throughput (Mbit/s) of four scenarios with training of one user



(b) Packet delay (ms) of four scenarios with training of one user

Figure 7: Mean values of throughput and packet delay of Verus and the model on channel trace #1 for network scenarios with one, two, three, and four users

6.4 Training with a Matching User Model

From the observations of the previous section and as highlighted in [18, 14], the most dominant factor of an accurate model is the match of the number of users in the training and evaluation phase. Therefore, in this section the model is evaluated in a multi-user environment, i.e., when several users with the same model compete for the channel capacity of the link. That is, the number of users in the model matches the number of users in the scenario in which the model is evaluated in.

The setup of the scenarios is as described in Section 6.1. The results are obtained by applying a quantization of 2 ms when generating the state transition matrix. All other parameters were set as stated in Table 1. Each simulation was repeated five times in order to obtain statistical confidence. The following scenarios are investigated for all five channel traces:

- Training with 2 users and evaluation with 2 users.

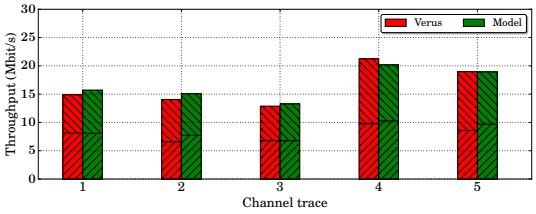
- Training with 3 users and evaluation with 3 users.
- Training with 4 users and evaluation with 4 users.

Figure 8 to 9 show the mean values of throughput and packet delay for the scenarios with two, three, and four users of Verus (red) and the model (green), respectively. The individual mean values of throughput and network delay per flow are shown as a stacked plot. Hence, the sum of all individual flows in the throughput figures denote the total achieved throughput over the given channel trace. It is important to mention that Verus does not always achieve the maximum available throughput of the channel due to the R parameter of 2. As a result, the total throughput of the model can be slightly higher in some channel traces and scenarios.

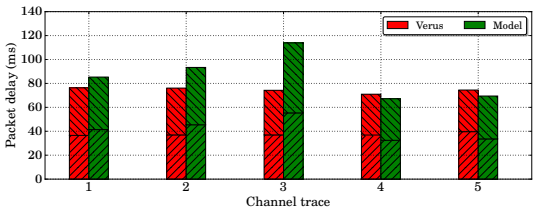
Figure 8 depicts the results for a scenario with a training of two users and an evaluation of the same number of users. It can be seen that the model shows marginally higher throughput for the channel traces #1 and #2 and consequently also a marginally higher packet delay per user (5 - 20% higher). This, however, is still in an acceptable range since higher throughput generally results in larger network delays. Likewise, in the channel trace #4 and #5, the throughput is marginally lower or equal compared to Verus and also yields smaller packet delays per user. As discussed earlier, this is based on the fact that the model in those scenarios experiences only minor fluctuations of the channel and hence resides in more dense regions of the state transition matrix. In addition, due to the steady channel conditions, self-transitions or transitions to neighboring states occur more frequently and are more precisely captured by the model than other states. In channel trace #3, the throughput shows comparable values to Verus, but the packet delay has increased by around 50% per flow. Due to the highly fluctuating channel characteristics in this channel trace, the model suffers from the different channel properties of the channel trace and hence a mismatch of the model.

Likewise, in a scenario where the the model was trained with samples from a network with three users and evaluated in a scenario with three users all executing the same model, the results revealed larger delays of around 20 - 40% per user on all 5 channel traces.

Figure 9 depicts the results of the mean values of throughput and network delay for a scenario where the model was trained with four users. The evaluation was done on a scenario with four users using the given model of four users. When compared to Verus, the model achieves slightly higher throughput among all flows and throughout the five channel traces. Due to the slightly higher throughput, the network delay is also increased as the protocol sends more packets than Verus. As a result, up to 30% higher network delays for the individual users on channel trace #1 and #2 are observed. On the contrary, the channel traces #4 and #5 show



(a) Throughput (Mbit/s) of two users per channel trace



(b) Packet delay (ms) of two users per channel trace

Figure 8: Mean values of throughput and packet delay of Verus and the model over five channel traces for a scenario with two users

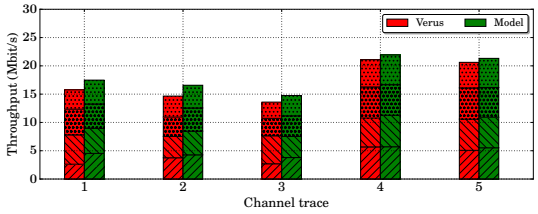
an almost equal network delay among all users even though the throughput is marginally higher compared to Verus. As stated earlier, this is due to the fact that these two channel traces show less bandwidth variations and hence are statistically better reflected by the model. As of channel trace #3, the network delay increased by more than twofold per user in comparison to Verus.

In conclusion it can be said that the model produces similar performance to Verus in most scenarios when the number of users in the training matches the number of users in the evaluation. Additionally, when observing the individual results of the throughput and network delay per user among all scenarios, it can be seen that the model achieves almost perfect fairness among users and hence shares the available bandwidth equally. This can be seen by the same length of the individual bars of the users per channel trace.

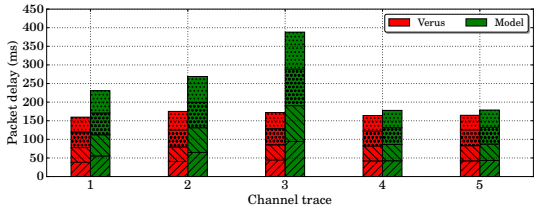
6.5 Training with a Combined User Model

In order to overcome the issues of a mismatch of the number of users in the training and evaluation phase and to create a model that suits scenarios where the number of users is unknown, the model in this section uses a combined set of training samples from one, two, three, and four users. This way, the model exhibits the statistical properties of scenarios with one to four users and is also evaluated in scenarios with one to four users.

The state transition matrix was generated with samples from the four scenarios. The four set of samples were combined to one dataset and serve as an input for the model. The evaluation of this model was executed in scenarios with one, two, three, and four users using the channel traces and parameters as described in Section 6.1 and Table 1. Again, a quantization of 2 ms

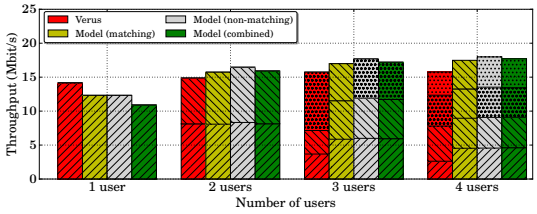


(a) Throughput (Mbit/s) of four users per channel trace

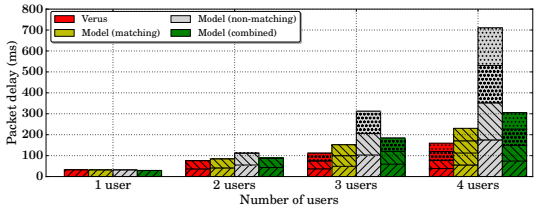


(b) Packet delay (ms) of four users per channel trace

Figure 9: Mean values of throughput and packet delay of Verus and the model over five channel traces for a scenario with four users



(a) Throughput (Mbit/s) of four scenarios with different number of users



(b) Packet delay (ms) of four scenarios with different number of users

Figure 10: Mean values of throughput and packet delay of Verus and the model on channel trace #1 for network scenarios with one, two, three, and four users

was used and all simulations were repeated five times. In order to limit the amount of results in this section, only the results of channel trace #1 are shown in the following. This channel trace serves as an example to highlight the results.

Figure 10 depicts the mean values of throughput and packet delay for the scenarios with one, two, three, and four users with a combined training set of one, two, three, and four users for channel trace #1. In addition to the results from this set of scenarios, the results from the matching user model (Section 6.4) and the non-matching user model (Section 6.3) for the same set of scenarios are shown in yellow and gray, respectively.

It can be seen that the performance of the model increased significantly in terms of network delay compared to the scenarios with a non-matching user model, especially in the scenario where four users are present. However, it still does not reach the network delays of the scenarios with the matching user model. Further, it can be observed that the throughput shows very similar results throughout all scenarios. That is, the model is slightly more conservative as in the non-matching scenarios and the state transition matrix with a combined set of users reflects the variety of scenarios with different numbers of users better. By means of that, combining samples from several sets of users is a suitable compromise when the number of users in the evaluation is unknown. As can be seen in the figure, this leads to an acceptable performance despite slightly higher delays compared to the scenario with a matching user model.

6.6 Evaluation with ON/OFF Traffic Model

Although the results of the previous section showed already reasonable results, this section evaluates the performance of Verus and the model in a more realistic scenario. In this scenario, the four users in the network are switched ON and OFF based on a stochastic process and share the same bottleneck link. That means, each user has independent traffic characteristics and is ON to transmit a certain number of bytes and OFF for specific time. Due to this, a scenario is created that represents a mixed number of users with specific transmit intervals and idle times.

The traffic model that describes the ON and OFF periods of the users was inspired by the FTP traffic model of [1] and uses the parameters as listed in Table 3. The file size implicitly defines the ON duration of the user and follows a truncated log-normal distribution. In order to suit the higher bandwidths of the channel traces, the parameters of the distribution are increased by a factor of two compared to the traffic model specifications in [1]. In addition, to achieve time-synchronized results[3] for both Verus and the model, the file size of the traffic model was converted to a time by dividing the file size by the mean throughput (∼4 Mbit/s) that was observed per user in the previous evaluations. This yields the mean time a user needs to complete the transfer of the file. For each ON period, a new connection of the protocol is initiated and results in overlapping transmit intervals among the users. Further, the time for the OFF period follows an exponential distribution and the mean was halved compared to the traffic model

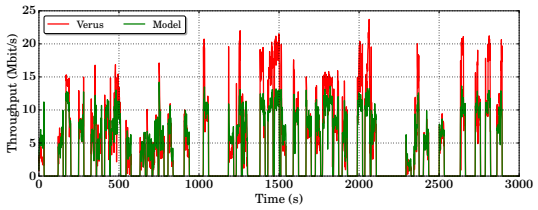
[3]Due to a slight difference in the throughput performance of both protocols, the file transfer completion times differ and hence trigger different starting times of the OFF periods.

specifications in [1]. This was done to have shorter idle periods of the individual users and to shorten the simulation time.

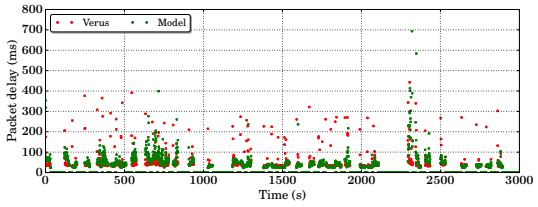
Component	Distribution	Parameters	PDF
File size	Truncated Lognormal	Mean: 4 Mbyte Variance: 3.3 Mbyte Maximum: 10 Mbyte	$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{\frac{-(\ln(x)-\mu)^2}{2\sigma^2}}$ for $x \geq 0$ mean $= e^{\mu+\frac{\sigma^2}{2}}$ var. $= (e^{2\mu+\sigma^2})(e^{\sigma^2}-1)$
OFF duration	Exponential	Mean: 60 s	$f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$ mean $= 1/\lambda$

Table 3: ON/OFF traffic model parameters

The state transition matrix for this scenario was generated as denoted in Section 6.5 with a combined set of samples from one, two, three, and four users with a quantization of 2 ms. Since the length of each individual channel trace from the previous section was limited to 300 s, the five channel traces were concatenated in this scenario to obtain a total length of 1500 s and a larger variety of channel conditions. The simulation, however, was run for 3000 s in order to capture more ON and OFF periods of the individual users. That is, the channel trace is repeated after 1500 s of simulation time from the beginning. In addition, the simulation of both protocols use the same sequence of random numbers per seed that describe the file size and the time for the OFF periods. This allows for two identical traffic models in both simulations and to compare Verus and the model when using the same seed.



(a) Throughput (Mbit/s) over time of user 1

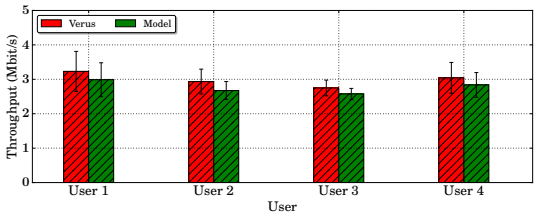


(b) Packet delay (ms) over time of user 1

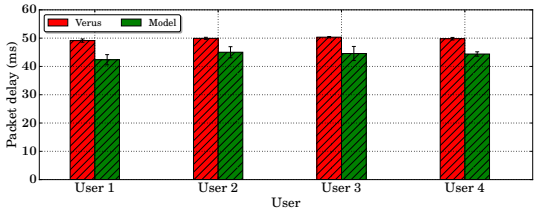
Figure 11: Throughput and packet delay over time of user 1 for Verus and the model in a scenario with ON/OFF traffic model.

Figure 11 depicts the throughput and the packet delay of user 1 over time for Verus and the model. It can be seen that both protocols achieve a very similar throughput and also almost equal packet delays. However, due to the slow-start phase of Verus, some peaks are visible in Figure 11b. Further, it can be observed that the maximum throughput among the different ON periods differs. This is mainly caused by a combination of the available bandwidth that is defined by the channel trace and by the number of users that are also active during these ON periods. As a result, the protocols show in some time instances higher throughput than in others. The results of the other three users show similar performance, but the ON and OFF periods differ due to the stochastic nature of the model.

Figure 12 depicts the mean values of throughput and network delay per user for Verus and the model and includes the 95% confidence intervals of the five independent simulations. It can be seen that the model achieves marginally lower throughput but the confidence intervals overlap with the confidence intervals of Verus. That is, the performance of the model is comparable to Verus among all four users. However, the network delay of the model is slightly lower compared to Verus. In total it can be observed that both protocols achieve almost perfect fairness among all users and scenarios.



(a) Throughput (Mbit/s) per user



(b) Packet delay (ms) per user

Figure 12: Mean values of throughput and network delay for Verus and the model in a scenario with a ON/OFF traffic model.

7. CONCLUSION

This paper presented a novel approach to stochastically model delay-based congestion control protocols. The objective of this model is to describe delay-based congestion control protocols using a two-dimensional discrete-time Markov model that can be generated from samples of a training set generated through simulations. The model presents a concise description of the delay-based congestion control protocol that bridges the gap in our understanding of delay-based protocols. Furthermore, the paper presented an implementation of the

model that can be used as a replacement for delay-based congestion control protocols. We evaluate our model against, Verus, a state-of-the-art delay-based congestion control protocol and demonstrate comparable performance across a variety of network and user settings. We find that the model simplifies the implementation complexity of delay-based congestion control protocols significantly without major performance degradations.

8. REFERENCES

[1] 3GPP2. cdma2000 Evaluation Methodology. Available at `http://www.3gpp2.org/public_html/specs/C.R1002-0_v1.0_041221.pdf`, 2004. Accessed: 2015-12-12.

[2] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan. PCP: Efficient Endpoint Congestion Control. In *NSDI*, 2006.

[3] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. *TCP Vegas: New techniques for congestion detection and avoidance*, volume 24. ACM, 1994.

[4] B. Briscoe and J. Manner. Byte and packet congestion notification, February 2014. RFC7141.

[5] C. Casetti, M. Gerla, S. Mascolo, M. Sanadidi, and R. Wang. TCP Westwood: end-to-end congestion control for wired/wireless networks. *Wireless Networks*, 8(5):467–479, 2002.

[6] M. Dong, Q. Li, D. Zarchy, B. Godfrey, and M. Schapira. PCC: Re-architecting Congestion Control for Consistent High Performance. *arXiv preprint arXiv:1409.7092*, 2014.

[7] Google. SPDY: An experimental protocol for a faster web. Available at `http://dev.chromium.org/spdy`, 2010. Accessed: 2015-09-06.

[8] Google. QUIC, a multiplexed stream transport over UDP. Available at `http://www.chromium.org/quic`, 2015. Accessed: 2015-09-01.

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

[10] J. Olsén. *Stochastic modeling and simulation of the TCP protocol*. PhD thesis, 2003.

[11] Riverbed Technology. OPNET Modeler 17.5. Available at `http://www.riverbed.com/products/steelcentral/opnet.html`.

[12] C. B. Samios and M. K. Vernon. Modeling the throughput of TCP Vegas. In *ACM SIGMETRICS Performance Evaluation Review*, volume 31, pages 71–81. ACM, 2003.

[13] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low extra delay background transport (ledbat), December 2012. RFC6817.

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

[15] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *Proc. of 25th IEEE International Conference on Computer Communications INFOCOM*, pages 1–12. IEEE, Apr. 2006.

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

[17] K. Winstein and H. Balakrishnan. TCP Ex Machina: Computer-generated Congestion Control. In *Proc. of the ACM SIGCOMM 2013 Conference*, 2013.

[18] K. Winstein, A. Sivaraman, H. Balakrishnan, et al. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proc. of the 10th USENIX conference on Networked Systems Design and Implementation*, pages 459–471, 2013.

[19] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg. Adaptive Congestion Control for Unpredictable Cellular Networks. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, London, UK, Aug. 17-21 2015.