Classification of Delay-based TCP Algorithms From Passive Traffic Measurements

Desta Haileselassie Hagos*, Paal E. Engelstad†, Anis Yazidi‡
*†University of Oslo, Department of Technology Systems, Kjeller, Norway
*†‡Oslo Metropolitan University, Department of Computer Science, Oslo, Norway
Email: *destahh@ifi.uio.no, {*desta.hagos, †paal.engelstad, ‡anis.yazidi}@oslomet.no

***Abstract*—Identifying the underlying TCP variant from passive measurements is important for several reasons, e.g., exploring security ramifications, traffic engineering in the Internet, etc. In this paper, we are interested in investigating the delay characteristics of widely used TCP algorithms that exploit queueing delay as a congestion signal. Hence, we present an effective TCP variant identification methodology from traffic measured passively by analyzing β, the multiplicative back-off factor to decrease the *cwnd* on a loss event, and the queueing delay values. We address how the β as a function of queueing delay varies and how the TCP variants of delay-based congestion control algorithms can be predicted both from passively measured traffic and real measurements over the Internet. We further employ a novel non-stationary time series approach from a stochastic nonparametric perspective using a two-sided Kolmogorov–Smirnov test to classify delay-based TCP algorithms based on the α, the rate at which a TCP sender's side *cwnd* grows per window of acknowledged packets, parameter. Through extensive experiments on emulated and realistic scenarios, we demonstrate that the data-driven classification techniques based on probabilistic models and Bayesian inference for optimal identification of the underlying delay-based TCP congestion algorithms give promising results. We show that our method can also be applied equally well to loss-based TCP variants.**

***Keywords*—TCP, Delay-based, LSTM, Bayesian, Classification, Kullback-Leibler divergence, Kolmogorov–Smirnov, Stochastic**

I. Introduction and Motivation

Transmission Control Protocol (TCP) is one of the dominant transport protocols that has played a great role in the exponential success of the Internet, network technologies and applications [17]. The majority of all Internet traffic all over the world today use TCP. TCP is a highly reliable end-to-end connection-oriented transport protocol designed to prevent excessive congestion on the Internet [17]. Note that congestion control in TCP was not part of the protocol initially until the first Internet congestion collapse was observed [17]. TCP controls congestion by aiming for fair sharing of the available network resources by the competing flows, using strategies empowered by TCP [17]. Congestion control algorithms provide a fundamental set of techniques critical for maintaining the robustness, efficiency, and stability of the global Internet. Since the specification of TCP [25], various end-to-end congestion control algorithms have been designed and implemented on a larger scale for the Internet with several enhancements. One category of the widely deployed variants ranging from TCP CUBIC [12], Reno [17], BIC [30], etc. where packet loss probability is an implicit signal for congestion in the underlying network are called loss-based TCP congestion control algorithms. Variants of this kind aggressively fill up the actual network buffers in order to achieve better throughput by ignoring queueing delay. However, this is challenging for the quality of latency-sensitive and bandwidth-intensive real-time media applications to achieve good performance when long-running flows also share a large bottleneck link buffers. Therefore, to address this challenging problem, delay-based TCP schemes that adopt packet queueing delay rather than a loss as congestion signals are introduced. With delay-based congestion control algorithms, allocating network resources across competing users can be attained by supporting both high network utilization and low queuing delay even when the buffer sizes are large. Detailed background on these categories of TCP variants is presented in Section II. In this paper, we are interested in investigating the delay characteristics of widely used TCP algorithms that exploit queueing delay as a congestion signal and demonstrating how an intermediate node can identify both loss-based and delay-based TCP variants from passively captured TCP traffic using *state-of-the-art* approaches. As explained below, inferring whether the underlying network is using *loss-based* or *delay-based* TCP congestion control algorithms has potential benefits for various reasons. Our work in this paper is mainly motivated by the following questions: *(i)* How well can we infer the most important TCP per-connection transmission states that determine a network condition (e.g., *Congestion Window (cwnd)*) from a passive traffic collected at an intermediate node of the network without having access to the sender? *(ii)* How can we identify the underlying delay-based TCP variant that the TCP client is using from passive measurements? *(iii)* What percentage of network users are using *delay-based* TCP variants? *(iv)* How do different implementations of TCP congestion control algorithms behave on the end-to-end variability of bandwidth, delay, different cross-traffic, Round-Trip Time (RTT)? etc.

Potenial opportunities and benefits: It is reasonable to ask: *Why is the identification of the underlying TCP flavors performed in an intermediate node from passive measurements important*? Some of the main reasons why passive estimation of TCP internal state variables in an intermediate node is important, for example, is when *(i)* We have no control over either end-host of communication so we can't launch active measurements from either host, *(ii)* The TCP active probes used in active measurements (such as *ping* messages) are blocked by firewalls etc. In addition to this passive measurements, unlike active measurements, do not introduce additional traffic into the network that can skew the results. For more details about the difference between *active* and *passive* measurement techniques, we refer the reader to our previous work [13]. There are myriad reasons we may want to use passive measurements to identify the underlying TCP flavors but in our paper, we will explore the above questions quantitatively from different perspectives as they apply to the problems of network congestion.

Operational benefit: We argue that uniquely inferring the underlying TCP congestion algorithm the client is using is useful for network operators to monitor if major content providers (e.g., *Google*, *Facebook*, *Netflix*, *Akamai* etc.) are manipulating their congestion windows in their servers to achieve more than their fair share of available bandwidth. Another scenario where network operators might find this information useful is if they have a path that they know is congested due to customer complaints, but the links using that path are not especially over-subscribed. In that case, details about the cwnd behaviour of all the users on that path might be helpful in trying to diagnose the cause, i.e., *are there users that are using aggressive congestion control algorithms which are unfair and affecting other user's available bandwidth*?

ISP benefit: Knowledge about the TCP stack in use in the endpoints is useful for operators of big ISP networks that do much traffic engineering who need to move traffic from oversubscribed links. It can also be used to diagnose TCP performance problems (e.g., to determine whether the sending application, the network or the receiving network stack are to blame for slow transmissions) in real-time. Another benefit might be to observe when large content providers implement their own custom congestion control behavior that does not match to one of the known congestion control algorithms.

Security ramifications: We believe it is also useful for exploring security threats. This is because if we are able to infer the TCP variant, we can also make some guessing on the implementation of the underlying operating system and search for vulnerabilities. This can tell us about the encryption at the end-system that can be used to tailor-made attacks.

Contributions. Our paper makes the following contributions.

- We identify the main challenges in investigating the delay characteristics of the widely used TCP algorithms.
- We demonstrate how the intermediate node (e.g., a network operator) can predict the *cwnd* size of delay-based TCP algorithms using *state-of-the-art* deep learning techniques.
- We examine a set of *state-of-the-art* techniques that are reasonably effective in classifying the underlying variants of delay-based TCP congestion control algorithms within flow from passive measurements based on the β parameter.
- We employ a novel non-stationary time series approach from a stochastic nonparametric perspective using a two-sided Kolmogorov–Smirnov test to classify delay-based TCP algorithms based on the α parameter.
- We are able to identify the widely used TCP variants with high accuracy and explore security ramifications.
- We compare our delay-based classification approach with recent *state-of-the-art* loss-based identification techniques.
- Finally, we show that the learned prediction model performs reasonably well by leveraging trained knowledge from the *emulated* network when it is applied and transferred on a *realistic* scenario setting.

II. Background

TCP congestion control strategies are broadly categorized into loss and delay sensitive schemes. Loss-based TCP congestion control algorithms consider packet loss as an implicit indication of congestion by the network. TCP variants of this kind attempt to fill the network buffers and hence they tend to induce large queueing delays when the buffer sizes are large. Unlike traditional loss-based approaches, delay-based TCP congestion control algorithms use the changes in queueing delay measurements as implicit feedback to congestion in the network. Delay-based congestion control algorithms attempt to avoid network congestion by monitoring the trend of network path's RTT information contained in packets. It is believed that variants of this kind achieve better average throughput by not filling buffers and maintaining full path utilization with low queueing and fair allocation of rates to flows [4, 28]. In order to properly allocate, share the underlying network resources, and ensure network queueing delay stays low, delay-based congestion control algorithms require knowledge of an accurate estimate of the network path's *BaseRTT* [21], usually defined as the smallest of all measured minimum RTTs of a segment in the absence of congestion. The following are the list of an end-to-end widely used delay-based congestion control algorithms on the Internet we use for our evaluation.

- **TCP Vegas** [4]: Vegas is a delay-based implementation of TCP congestion control algorithm motivated by the studies [18] and [27]. Vegas's congestion detection technique depends on the accurate estimation of *BaseRTT* [4]. Hence, if the estimated value of *BaseRTT* is too small, then it's throughput will stay below the available bandwidth; however, if the estimated value for *BaseRTT* is too large, then it will overrun the connection [4]. As shown in Equation 1, Vegas computes the *expected* throughput of the connection as the ratio of the current window size and *BaseRTT*. The main idea of Vegas behind Equations 1 and 2 is that when the network is not congested, the actual flow rate will be close to the expected flow rate. However, if the network is congested, the expected flow rate will be greater than the actual flow rate.

$$Expected = \frac{WindowSize}{BaseRTT} \tag{1}$$

$$Actual = \frac{WindowSize}{RTT} \tag{2}$$

To estimate the available bandwidth and congestion state of the network, TCP Vegas compares the actual sending rate and evaluates the difference, *Diff*, between the *estimated* throughput and the current *actual* throughput computed as shown in Equation 3 and updates the *cwnd* accordingly.

$$Diff = Exptected - Actual \tag{3}$$

By definition *Diff* is a non-negative since *Actual*>*Expected* implies that we need to change *BaseRTT* to the latest sampled RTT. To adjust the congestion window size, TCP Vegas uses *two* threshold values α and β where $0 \leq \alpha < \beta$ [4]. Depending on this difference as shown in Figure 1 and Equation 4, if *Diff*$<\alpha$, Vegas increases the *cwnd* size linearly until the next RTT, and when *Diff*$>\beta$, then Vegas reduces the *cwnd* linearly until the next RTT. However, Vegas leaves the *cwnd* size unchanged when $\alpha <$ *Diff* $< \beta$.

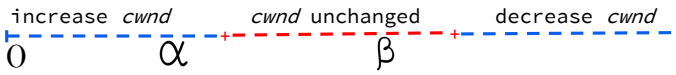


Fig. 1: TCP Vegas throughput thresholds

$$cwnd = \begin{cases} cwnd + 1 & \text{If } Diff < \alpha \\ cwnd - 1 & \text{If } Diff > \beta \\ cwnd & \text{Otherwise} \end{cases} \quad (4)$$

- **TCP Veno** [11]: Veno adopts the same methodology as TCP Reno [17] in determining the congestion window size in the network. But Veno uses the delay information of TCP-Vegas [4] to further differentiate non-congestion packet losses when RTT varies greatly by estimating the backlogged packets in the buffer similar to TCP Vegas. If the number of backlogged packets in the buffer is below a certain threshold, it is a strong indication of random loss. However, if packet loss is detected when the connection is in the congestive state, TCP Veno uses the standard TCP Reno Additive Increase and Multiplicative Decrease (AIMD) scheme to reduce the *cwnd* during its congestion avoidance mode. TCP Veno sets β factor to 0.8 when the queueing delay is small. However, when the queueing delay is high, TCP Veno sets β to 0.5.

For comparison reasons we also consider the following most widely used loss-based TCP congestion control algorithms.

- **TCP Reno** [17]: Reno is one of the most predominant implementations of loss-based TCP variants which employs a conservative linear growth function for increasing the *cwnd* by one segment for each received ACK and multiplicative decrease function on encountering a packet loss per RTT [5]. Its β value is 0.5 which corresponds to reducing the window by 50% during a loss event as shown in Equation 5.

$$cwnd = \begin{cases} cwnd + 1 & \text{Slow start phase} \\ cwnd + \frac{1}{cwnd} & \text{Congestion avoidance} \\ \frac{cwnd}{2} & \text{If packet is lost} \end{cases} \quad (5)$$

- **TCP CUBIC** [12]: CUBIC is the default congestion control algorithm as part of the Linux kernel distribution configurations from version 2.6.19 onwards. It modifies the linear window growth function of existing TCP standards to be governed by a *cubic function* in order to improve the scalability of TCP over fast and long distance networks. CUBIC decreases the *cwnd* by a factor of β whenever it detects that a segment was lost. And, it increases towards a target congestion window size (W) when in-order segments are acknowledged where W is defined as follows:

$$W_{cubic}{}^{(t)} = |C(t-K)|^3 + W_{max} \quad (6)$$

where W_{max} is the window size reached before the last packet loss event, C is a fixed scaling constant that determines the aggressiveness of window growth, t is the elapsed time from the last window reduction measured after the fast recovery, and K is defined by the following function:

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \quad (7)$$

where β is a constant back-off factor of CUBIC [12] applied for window reduction at the time of a TCP packet loss event. The β value of CUBIC is 0.7 which corresponds to reducing the window by 30% during a TCP packet loss event [12] and can be calculated as per Equations 6 and 7.

Roadmap: The rest of this paper is organized as follows. Next, in Section III, we discuss the related work in the literature considered as a *state-of-the-art*. In Section IV, we describe an overview of our controlled experimental setup for the evaluation. Section V presents approaches to our classification models in detail. The experimental results and discussion are presented in Section VI. Finally, Section VII concludes the paper and outlines directions of research for future extensions.

III. Related work

TCP is one of the key protocols of today's Internet Protocol (IP) suite and its performance analysis has been extensively studied in the computer networking community [24]. Many research studies have also analyzed the underlying TCP congestion control algorithms as we have already discussed the most relevant works in Section II. There are many different TCP variants widely in use, and each variant uses a specific end-to-end congestion control algorithm to avoid congestion, while also attempting to share the underlying network capacity equally among the competing users. However, we believe that there is very little work on the identification of the underlying delay-based TCP congestion control algorithms from passive measurements. The work in [22] proposes a *cluster analysis-based* method that aims a router to identify between two versions of TCP algorithms. This method was meant to be utilized in real-time applications to handle network traffic routing policies. It performs RTT and *cwnd* estimation in order to infer a group of traffic characteristics from the flow [22]. These characteristics are then clustered into two groups by applying a hierarchical clustering technique. The authors show that only 2 out of 14 TCP congestion algorithms that are implemented in Linux can be identified based on their method [22]. Another related work [31] presents an *active* measurement technique to identify a diverse set of known congestion control algorithms. However, our work in this paper relies on a *passive* measurement technique. In a closely related previous work [15], we presented a machine learning-based approach to identify the underlying traditional *loss-based* TCP variants which yield accuracies of 93.51% and 95% on emulated and realistic scenarios respectively. The *cwnd* prediction performance result of the loss-based variants across different scenario settings is presented in Table I.

TABLE I: cwnd prediction accuracy of *loss-based* TCP variants under an *emulated* and *realistic* settings [14, 15]

	Emulated Setting		*Realistic Setting*	
TCP Algorithms	**RMSE**	**MAPE** (%)	**RMSE**	**MAPE** (%)
CUBIC	5.839	6.953	3.522	4.738
Reno	3.511	3.140	3.396	5.197

This was achieved by analyzing the β value by averaging out the window size of *loss-based* algorithms every time a peak is detected so that the computation of the multiplicative decrease factor is not done only on a *slow start* phase. However, this work doesn't address how the β as a function of queueing delay varies and how the TCP variants of delay-based congestion control algorithms can be predicted both from passively measured traffic and real measurements over the Internet. By design, unlike loss-based algorithms, the β value of delay-based congestion control algorithms is not fixed which makes it fundamentally challenging to predict the TCP variant from passive traffic when there is variability in delay. Hence,

in this paper, we want to substantially address this problem by building a two-dimensional space model and see if the β is dependent on queueing delay or not. This helps us to expand our previous method [15] to address bigger cases covering both loss-based and delay-based TCP congestion control algorithms.

IV. Evaluation Methodology

A. Experimental Setup

Our evaluation experiments are carried out using a cluster of HPC machines based upon the GNU/Linux operating system running a modified version of the *4.15.0-39-generic* kernel release. The prediction model is performed on an NVIDIA Tesla K80 GPU accelerator computing with the following characteristics: Intel(R) Xeon(R) CPU E5-2670 v3 @2.30GHz, 64 CPU processors, 128 GB RAM, 12 CPU cores running under Linux 64-bit. All nodes in the cluster are connected to a low latency 56 Gbit/s Infiniband, gigabit Ethernet and have access to 600 TiB of BeeGFS parallel file system storage.

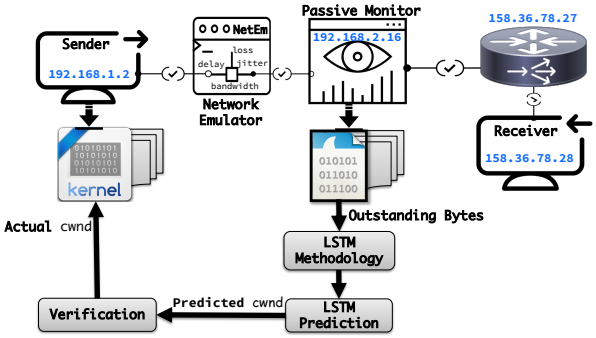


Fig. 2: LSTM Methodology for *cwnd* prediction

B. Validation Experiment

We have conducted a controlled experiment both on simulated environments and realistic scenario settings.

Emulated Setting: We construct an experimental setup shown in Figure 2 where we generate the training data and predict the *cwnd* from a passively captured traffic using state-of-the-art deep learning approaches. To evaluate the prediction model and perform our analysis on both the emulated and realistic network conditions, we have generated our own training dataset. In order to capture all sessions on the network when the client and server are sending TCP packets and measure the TCP data packets from both directions, we have used the fully controlled experimental setup shown in Figure 2. The background TCP stream traffic for all our experiments are generated using the *iperf* [9] traffic generator on an emulated LAN link where we run each TCP variant by adding a variation of the emulation parameters *bandwidth (in Mbit)*, *delay (in ms)*, *jitter (in ms)* and *packet loss (%)* within a flow. During a single TCP flow of our experiment, the parameters *bandwidth*, and *delay* are *constant* with a *uniform* distribution. However, since we have the *jitter* given as an average, its distribution is *normal*. The issues of cross-traffic variability and verification of the popular Linux-based network emulator we used, *Network Emulator (NetEm)* [16], are thoroughly addressed in our previous work [13].

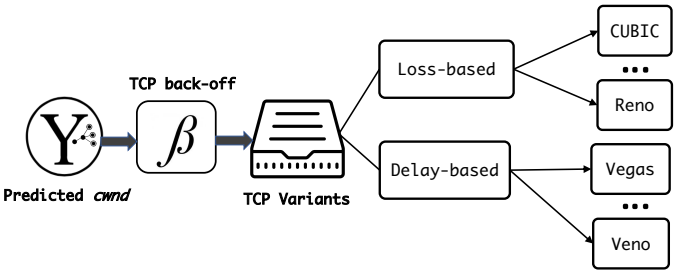


Fig. 3: Methodology for TCP Variant classification

Realistic Setting: In addition to the simulation validation described above, we have also evaluated our experiments over real-world Internet paths using the setup shown in Figure 4 so that we can further validate our results presented in Figure 9 against other scenario settings. This helps us to carefully test how well our deep learning-based cwnd prediction model using an emulated network works by conducting a series of controlled experiments in a realistic setting. In this way, we can justify and guarantee how our model could predict the development of a *cwnd* pattern and the TCP variant used with other realistic network traffic scenarios captured from the Internet. To this end, we created a realistic experimental testbed where we experiment by running our resources on Google Cloud platform nodes across the Internet as shown in Figure 4. In order to create a realistic TCP session, we uploaded a massive image file to Google Cloud platform site so that we have a full control of the underlying TCP variant on the sending node and at the same time run a *tcpdump* in the background and capture all sessions on the network when the client and server are sending TCP packets. Next, we filtered out the receiving host where we send the TCP traffic to. Finally, we calculated the number of outstanding bytes obtained from the captured network traffic and run it through our learning model to predict the development of the TCP *cwnd*. Since we have full control of the sending node, we can track the system-wide TCP state of every packet that is sent and received from the kernel to verify our model's prediction accuracy against the ground truth by matching with the actual sending TCP states using the methodology shown in Figure 2. As it is shown in Figure 10, we found out that our model could be performing very well with small prediction errors when we apply it to real-world scenario settings too. The final cwnd sawtooth pattern prediction performance comparison between the emulated and realistic settings is presented in Table V.

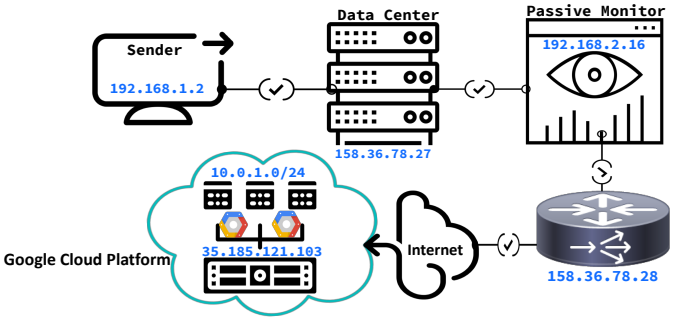


Fig. 4: Realistic Scenario Setup

V. Our Approaches

This section presents the concepts and approaches to the underlying TCP variant classification process.

A. Passive cwnd Prediction

For this task, we are interested in the capabilities and potentials of *Recurrent Neural Networks (RNN)* models for implementing our passive *cwnd* prediction model for TCP. Hence, we have explored an approach to investigate and explore in detail on how an intermediate node (e.g., a network operator) can identify the transmission state of both loss-based and delay-based TCP congestion control algorithms associated with a passively monitored TCP traffic using *Long Short-Term Memory (LSTM)*-based RNN architecture. In TCP, the *cwnd* is one of the main factors that determine the number of bytes that can be outstanding at any time. Hence, we assume that using the observed outstanding sequence of unacknowledged bytes on the network seen at any point in time in the lifetime of the connection as an estimate of the sending TCP's *cwnd* from *tcptrace* [23] when there is variability of *bandwidth*, *delay*, *jitter* and *loss* is a better approach to estimate the *cwnd* and how fast the recovery is. We measure the *cwnd* for the end-to-end path between the sender and the receiver basing our inference on the total amount of *outstanding* bytes and run it through our learning model to predict the development of the TCP *cwnd* and it's variant.

Implementation details: Our methodology of the classification process is depicted in Figure 3. We implemented our passive *cwnd* prediction model in *Python* using the *Keras* deep learning framework with *Google's TensorFlow* backend [1] running on NVIDIA Tesla K80 GPU where we apply an LSTM-based architecture trained over multiple epochs by taking the *cwnd* samples as values in time-series. As shown in Figure 2 at each time-step of t, as a learning process, the LSTM model takes an entire array of the outstanding bytes matching based on *timestamps* captured on the intermediate monitoring point between the sender and receiver as an input feature vector indexed by *timestamps*. We propagate the input to the model through a multilayer LSTM cell followed by a dense layer of 15-dimensional hidden states with Rectified Linear Unit (ReLU) activation function for the different layers that generates an output of a sequence dimensional vector of predicted *cwnd* of the same size indexed by *timestamps*. Our LSTM network is trained using the Truncated Back Propagation Through Time (TBPTT) training algorithm for modern RNNs applied to sequence prediction problems [26]. We used this training algorithm to minimize LSTM's total prediction error between the expected output and the predicted output for a given input of the measured *cwnd* time-series. We trained our LSTM-based learning algorithm without the knowledge of the input features from the TCP sender-side during the learning phase. We learn the model from the training data and then finally predict the test labels from the testing instances on all variations of the emulation parameters. In order to train our prediction model more quickly, and get a more stable and robust to changes *cwnd* estimation model, we have applied one of the most effective optimization algorithms in the deep learning community, the *Adam* stochastic algorithm [19] with an initial *learning rate* of *0.001* and *exponential decay rates* of the first (β_1) and second (β_2) moments set to 0.9 and 0.999 respectively. Totally, all of our configurations were trained for a maximum of 100 epochs with the mini-batch size of 256 samples. We further optimize a wide range of important optimal hyperparameters related to the neural network topology to improve the performance of our prediction model. In order to train and test our prediction model, we employed every experiment with a ratio of *60%* training, *40%* testing split and a *5*-fold cross-validation into one learning model.

Why did we use RNN models? As explained above, the *cwnd* is a TCP per-connection state internal variable, stored in the memory of the TCP sender, relevant to congestion control. However, since the value of the *cwnd* is not contained in the TCP header – trying to predict this value somewhere other than at the TCP sending node is fundamentally challenging. In our case, let's consider a situation where a network model is trained for a specific intermediate node which has been trained for a specific bandwidth, background load, multiplexing rate, and a multitude of different router conditions, can predict well for exactly this node. Hence, we want a model that is able to train in one scenario setting and apply it as a pre-training on another setting by leveraging trained knowledge. As it is presented in Section IV, this paper proofs that it makes sense in principle to use learning algorithms for TCP state predictions and this is the reason why we use RNN approach for the passive *cwnd* prediction.

B. TCP variant classification based on the β parameter

K-Nearest Neighbor (KNN): The first approach we used to identify the underlying TCP variants is a distance metrics using KNN machine learning classification algorithm [6]. Given an input feature vector of TCP protocols in an n-dimensional Euclidean space $\mathbb{R}^n$ with a set of back-off parameters and queueing delay instances, $\{\hat{\beta}_i, \widehat{diff}_i\} \in \mathbb{R}^n$, training samples of the form $(\{x_i, y_i\}, x_i \in \mathbb{R}^n)$, we want to classify a new TCP protocol, P, by finding the value of $\{\hat{\beta}_i, \widehat{diff}_i\}$ that is nearest to P. For the estimation of $\widehat{diff}$ in our evaluations, we applied the formula proposed in TCP Vegas as shown in Equation 3.

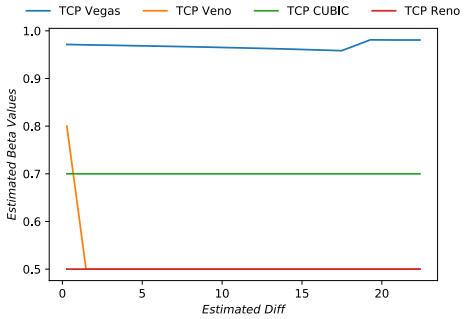


Fig. 5: KNN Prediction of TCP Variants

As shown in Figure 5, our classifier model fits reasonably well with high accuracy. However, we believe that this approach has a limitation in classifying TCP Reno [17] and TCP Veno [11] when the queueing delay is high. For example, if we have many $\hat{\beta}$ points of TCP Veno in a two-dimensional space with low $\widehat{diff}$ that means we will have more $\hat{\beta}$ values with one cluster of 0.8. How do we ensure that the $\widehat{diff}$ is low and how do we tell the exact difference between TCP Veno and Reno? Hence, to avoid this shortcoming, we proposed the following methods.

Kullback-Leibler (KL) Divergence: Before fitting our data into the beta distribution family, we wanted to answer the question: *How do we optimally choose the positive shape parameters, α and β, of beta distribution given in Equation 9?* Hence, we use the KL divergence [20] to find the fitting parameters in the beta distribution. KL, in statistics, information theory and pattern recognition, is a well-known distance measure between two probability distribution measures $p(y)$ and $q(y)$ defined as follows:

$$D(q\|p) := \mathbb{E}_{y\sim q(y)}\left[\log\frac{q(y)}{p(y)}\right] = \int q(y)\log\frac{q(y)}{p(y)}dy \quad (8)$$

In general, the KL divergence is only defined if $q(y) > 0$ for any value of y such that $p(y) > 0$.

Beta Distribution: As shown in Figure 8, the beta distribution is the best fitting model for the problem we address in this paper for all the TCP protocols except TCP Veno [11]. Beta distribution, parametrized by two positive shape parameters, denoted by α and β, is appropriate for representing the uncertainty of a continuous probability distribution and is defined by:

$$f(K|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}K^{\alpha-1}(1-K)^{\beta-1} \quad (9)$$

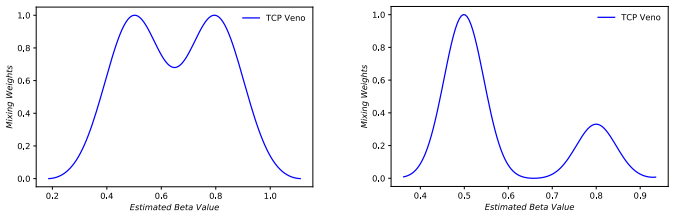
where $K \in [0,1]$ and it represents the support of the probability distribution, $\Gamma(\cdot)$ is the gamma function defined as: $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$.

For TCP Veno, as depicted in Figure 8(a), we used a *sigmoid*-based function and the reason why we applied *sigmoid* function for TCP Veno is because say that we have 1 million $\hat{\beta}$ points with high $\widehat{diff}$ value (i.e., when the connection is on a congestive state) and only 1 $\hat{\beta}$ point with low $\widehat{diff}$ value. If we compare this with TCP Reno [17] which has few points with a fixed $\hat{\beta}$ value of around 0.5, the inaccuracy might be a little higher when we measure it with other metrics. When we run our experiment on the Internet, we can't decide how many $\hat{\beta}$ points we get because it depends on the underlying network. As we can see it on Figures 6 and 7, we have one $\hat{\beta}$ point for Veno [11] and we believe this is completely realistic. Because this means we have one class (cluster) of 1 $\hat{\beta}$ point with low $\widehat{diff}$ and another class of many $\hat{\beta}$ points with high $\widehat{diff}$ and it is possible to classify the protocols based on these classes. We built our model in such a way that we don't want the sender or network change anything with the TCP parameter values (i.e., α and β) that control increase and decrease ratios of the *cwnd*. We simply want to observe things passively from an intermediate node between the sender and the receiver. Therefore, to tell the exact difference between the low $\widehat{diff}$ and high $\widehat{diff}$, we have to statistically measure how close we are to the border between the low and high $\widehat{diff}$. For example, when we are around a $\widehat{diff}$ threshold of 3, the $\widehat{diff}$ will not count much but the weight of the $\hat{\beta}$ value does. This way, we can tell the difference between the protocols TCP Veno and TCP Reno by running a *sigmoid*-based function on the border between low and high $\widehat{diff}$ values.

Mixture Distributions: For TCP Veno, we applied a beta mixture distribution model in a classification setting defined as follows:

$$f_1(\hat{\beta}_i, \widehat{diff}_i) = (1-\lambda_1(\widehat{diff}))d_1(\hat{\beta}_i) + \lambda_2(\widehat{diff})d_2(\hat{\beta}_i) \quad (10)$$

where λ_i being the mixing weights (density) of the *sigmoid* function that depends on the value of $\widehat{diff}$, $\sum_i \lambda_i = 1$, d_1 and d_2 are two different distributions. First, we pick a distribution of TCP Veno with probabilities given by the mixing weight, λ and $\widehat{diff}$, then we generate one observation according to the selected distribution as shown in Figure 11. Intuitively, the *sigmoid* of TCP Veno should be the weights of the two peaks of the $\hat{\beta}_i$ values, i.e., a beta distribution centered around 0.5 and a beta distribution centered around 0.8. For the other TCP protocols, we applied a beta distribution of the form $f_i(\hat{\beta}_i)$. We have experimented with different beta mixture distributions and finally, we have verified that our model yields reasonably good results. Sample beta mixture distributions of TCP Veno under different values are shown below in Figure 11.



(a) $\widehat{diff}$=3 (b) $\widehat{diff}$=4

Fig. 11: Mixture distributions of TCP Veno [11].

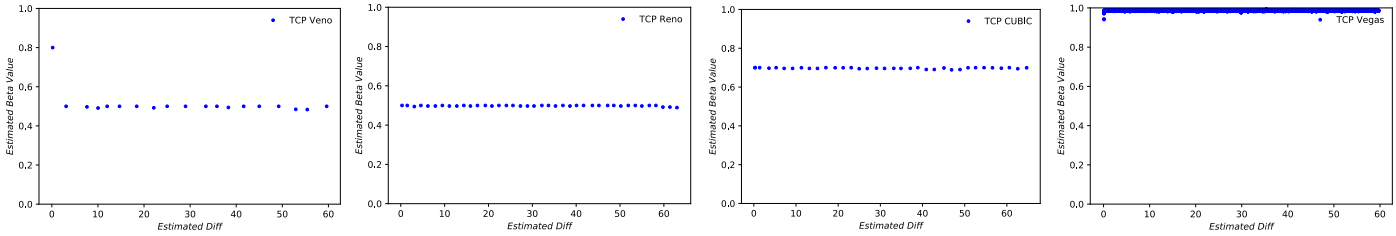
Bayesian Inference: Using the data generated from the beta distribution, we built a Bayesian inference approach to machine learning by constructing a set of observations $O_{1:N} = \{O_1, O_2, O_3, \ldots, O_N\}$ in which each element O_i represents a different set of observations of $\hat{\beta}_i$ and $\widehat{diff}_i$ of each TCP variant, V that is obtained from the beta distribution model as $f_i(\hat{\beta}_i, \widehat{diff}_i)$. As shown in Equation 12, the normalization factor is the sum of the data.

$$P(V=V_i|O_i) \propto P(V=V_i)\prod_{i=1}^{N} P(O_i|V=V_i) \quad (11)$$

From the law of probability theory, we know that:

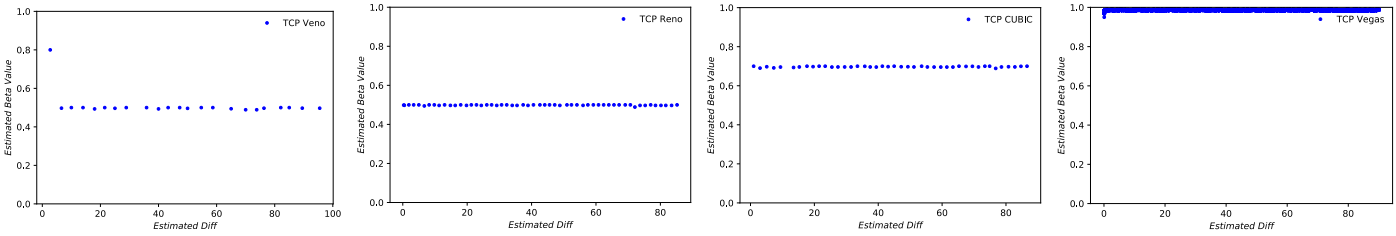
$$\sum_{V_i} P(V=V_i|\{O_1,O_2,O_3,\ldots,O_N\} = 1 \qquad V=\{V_1,V_2,V_3,V_4,\ldots,V_N\} \quad (12)$$

where $V_1 = Veno$, $V_2 = Reno$, $V_3 = CUBIC$ and $V_4 = Vegas$. For every V_i, the $argmax()$ of these equations retrieves the index of the highest likelihood of the probability vector. In the absence of a priori detailed domain knowledge about the TCP protocols, from a Bayesian inference perspective we believe all TCP variants will have the same probability and hence, $P(Veno) = P(Reno) = P(CUBIC) = P(Vegas)$. Using Equations 11 and 12, we are able to perform the Bayesian inference and the results we obtained from both emulated and realistic scenario settings are presented as follows.



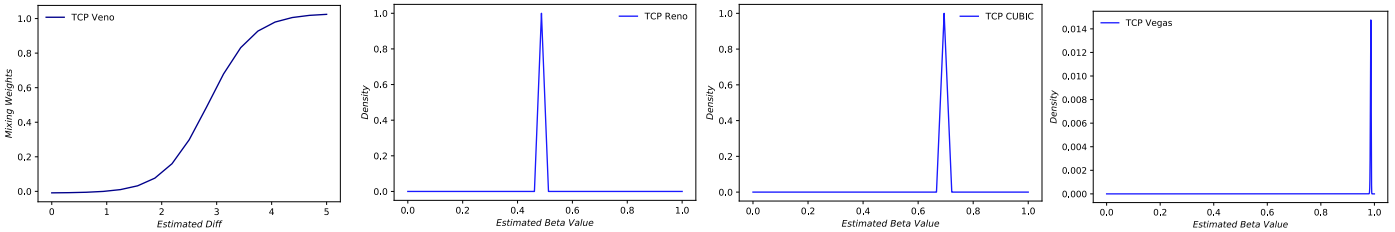
(a) (b) (c) (d)

Fig. 6: Beta analysis in an *emulated* setting. **(a)** Veno [11], **(b)** Reno [17], **(c)** CUBIC [12], **(d)** Vegas [4].



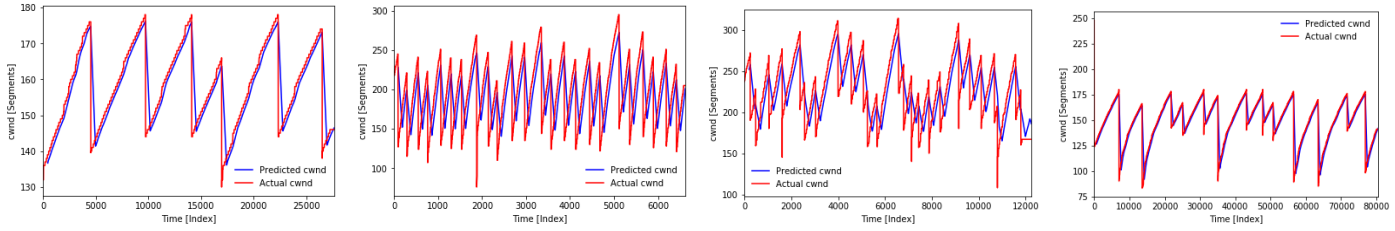
(a) (b) (c) (d)

Fig. 7: Beta analysis in a *realistic* setting. **(a)** Veno [11], **(b)** Reno [17], **(c)** CUBIC [12], **(d)** Vegas [4].



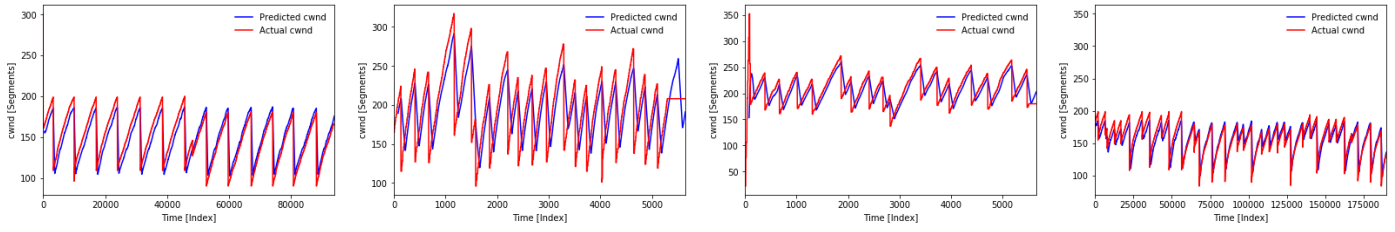
(a) Sigmoid analysis of TCP Veno [11] (b) (c) (d)

Fig. 8: Sigmoid analysis and beta distributions. **(a)** Veno [11], **(b)** Reno [17], **(c)** CUBIC [12], **(d)** Vegas [4].



(a) (b) (c) (d)

Fig. 9: TCP *cwnd* Prediction results of an *emulated* setting. **(a)** Veno [11], **(b)** Reno [17], **(c)** CUBIC [12], **(d)** Vegas [4].



(a) (b) (c) (d)

Fig. 10: TCP *cwnd* Prediction results of a *realistic* setting. **(a)** Veno [11], **(b)** Reno [17], **(c)** CUBIC [12], **(d)** Vegas [4].

Emulated setting

- When the ground truth is TCP Veno, $P(V = Veno|O_1, O_2, O_3, O_4)$ gives a probability estimation vector of (**46.28**, 38.93, 14.34, 0.45) and from this *46.28* maximizes the probability that this is being classified as V_1 (Veno).
- When the ground truth is TCP Reno, $P(V = Reno|O_1, O_2, O_3, O_4)$ gives a probability estimation vector of (35.25, **49.81**, 14.57, 0.36) and from this *49.81* maximizes the probability that this is being classified as V_2 (Reno).
- When the ground truth is CUBIC, $P(V = CUBIC|O_1, O_2, O_3, O_4)$ gives an estimation vector of (10.13, 9.02, **71.83**, 9.02) and from this *71.83* maximizes the probability that this is being classified as V_3 (CUBIC).
- When the ground truth is Vegas, $P(V = Vegas|O_1, O_2, O_3, O_4)$ gives an estimation vector of (31.85, 0.28, 10.79, **57.08**) and from this *57.08* maximizes the probability that this is being classified as V_4 (Vegas).

Realistic setting

- When the ground truth is TCP Veno, $P(V = Veno|O_1, O_2, O_3, O_4)$ gives a probability estimation vector of (**46.83**, 39.4, 13.44, 0.34) and from this *46.83* maximizes the probability that this is being classified as V_1 (Veno).
- When the ground truth is TCP Reno, $P(V = Reno|O_1, O_2, O_3, O_4)$ gives a probability estimation vector of (30.99, **52.05**, 16.44, 0.52) and from this *52.05* maximizes the probability that this is being classified as V_2 (Reno).
- When the ground truth is CUBIC, $P(V = CUBIC|O_1, O_2, O_3, O_4)$ gives an estimation vector of (10.69, 9.53, **70.25**, 9.53) and from this *70.25* maximizes the probability that this is being classified as V_3 (CUBIC).
- When the ground truth is Vegas, $P(V = Vegas|O_1, O_2, O_3, O_4)$ gives an estimation vector of (32.18, 0.39, 11.73, **55.7**) and from this *55.7* maximizes the probability that this is being classified as V_4 (Vegas).

C. TCP variant classification based on the α parameter

In our classification task of the underlying TCP algorithms, in contrast to the typical *increase-by-one decrease-to-half* scheme of TCP to adjust *cwnd* growth, we consider the β and α parameters that control the increase and decrease ratios of *cwnd*. This means, the *cwnd* size is increased by α per window of acknowledged packets in the congestion avoidance state in response to every RTT and it is decreased to β times its current value when there is congestion. Classifying the underlying TCP variant using the β parameter with different approaches is discussed above in detail. Here, we will use the α parameter for the same task by employing a novel non-stationary time series approach from a stochastic nonparametric perspective. We believe this approach is appealing because the changing rate of the *cwnd* size can be modeled as a stochastic process [2, 3]. This method is ensuring to work properly because of the quasi-stationary properties that every TCP protocol has as it could be easily observed from Figure 12(a) and Figure 12(b) where the statistical behavior of the signal remains almost unaltered and note that the distribution in all of the scenarios is pretty consistent by maintaining the same property. We use the two-sided Kolmogorov-Smirnov (KS) test which is a nonparametric statistical test[1] for comparing two empirical cumulative distribution functions (ECDFs) [10]. The KS statistic for a given cumulative distribution function (CDF) *F(x)* is given as shown in Equation 13.

$$D_n(F_n, G_n) = \sup_x |F_n(x) - G_n(x)| \tag{13}$$

where $\sup_x$ is the supremum of the set of distances over the given distributions, F and G are two ECDFs.

Our approach is to first estimate the probability distribution function (PDF) over categories in our classification task of each given TCP protocol as shown in Figure 12 and then estimate the 95% confidence interval for the distributions using bootstrap technique [7, 8] so that we can measure how certain we are about the predictions of the underlying TCP variant when its estimated α value changes frequently by comparing the uncertainty measure of the estimated PDF. We could also show the corresponding standard CDF for each value, but due to the limited space in this paper, here we present only the empirical PDF estimations. As we can see from Table IV, the stochastic confusion matrix has two values on both emulated and realistic settings. The first value compares the maximum difference between the ECDFs and chooses the protocol with the minimum distance that minimizes the probability of the $\log(p\text{-}value)$ as shown in Equation 14.

$$\gamma = \arg\min_{\gamma^\alpha} \int_\omega \left| \hat{P}_\gamma(\alpha) - \hat{P}_y(\alpha) \right| d\alpha \tag{14}$$

where γ^α represents the set of TCP protocols, ω represents all the possible values of α, $\hat{P}_\gamma(\alpha)$ represents the empirical probability of a given TCP protocol (γ), $\hat{P}_y(\alpha)$ represents the estimated probability of α in dataset y. Whereas the second value compares the KS test values by maximizing the estimated PDF of each distribution using $\log(p\text{-}value)$ of the bootstrap test for a given distribution as shown in Equation 15.

$$\gamma = \arg\max_{\gamma \in \{V_i\}} \log P[D_n(\gamma^C, y)], \quad i \in \{1, 2, 3\} \tag{15}$$

where y is another sampled time series of length M used to classify the protocols so that we don't end up using the same time series and γ^C are the candidate TCP protocols, V_1=*Veno*, V_2=*Vegas*, and V_3=*CUBIC*, from which the PDF were initially estimated. Next, we calculate the distribution function using the raw time series data of each variant and compare it against the stochastic template of each TCP protocol using the KS test value whose result is presented in $\log(p\text{-}value)$. Finally, we choose the underlying TCP protocol whose $\log(p\text{-}value)$ bootstrap test is higher. In Figure 12, if the α interval between 100 and 160 have too high values, then the TCP variant is Veno. Otherwise, if the value is too low between these intervals, the protocol will be identified as CUBIC. Our method is robust enough to identify each underlying protocol without the prior domain knowledge of the internal characteristics of each TCP variant. As it is shown in Figure 12(a) and Figure 12(b), it is clear that the model performs well in terms of identifying the underlying TCP variants when applied both on an emulated and realistic scenario settings.

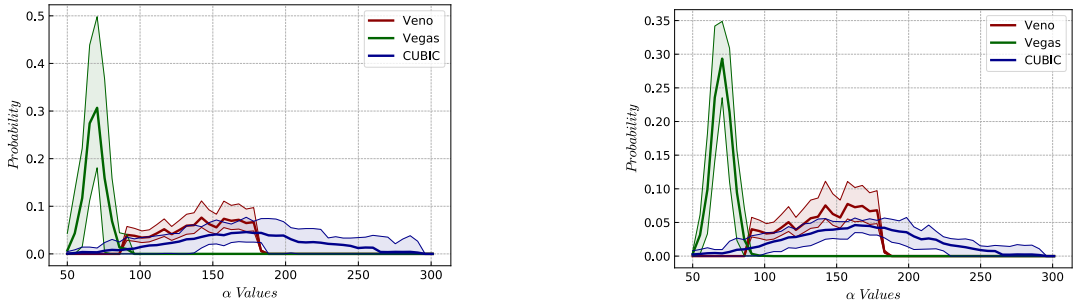
[1] i.e., it does not assume a specific form of the distributions

TABLE II: $\widehat{diff}$ values performances

	Low-$\widehat{diff}$				High-$\widehat{diff}$			
Emulated Setting								
	Precision	Recall	F1-score	Support	Precision	Recall	F1-score	Support
Vegas	1.00	0.92	0.96	13	1.00	0.91	0.95	11
CUBIC	1.00	1.00	1.00	10	0.92	1.00	0.96	11
Reno	0.90	1.00	0.95	9	0.90	1.00	0.95	9
Veno	1.00	1.00	1.00	8	1.00	0.89	0.94	9
Avg/Total	0.98	0.97	0.98	40	0.95	0.95	0.95	40
Accuracy	*97.5%*				*95%*			
Realistic Setting								
	Low-$\widehat{diff}$				High-$\widehat{diff}$			
	Precision	Recall	F1-score	Support	Precision	Recall	F1-score	Support
Vegas	1.00	0.93	0.96	12	0.92	0.92	0.92	11
CUBIC	1.00	1.00	1.00	11	1.00	1.00	1.00	10
Reno	0.92	1.00	0.96	9	0.91	1.00	0.95	9
Veno	1.00	1.00	1.00	8	1.00	0.91	0.95	10
Avg/Total	0.98	0.98	0.98	40	0.96	0.95	0.95	40
Accuracy	*97.83%*				*95.46%*			

TABLE III: $\widehat{diff}$ values confusion matrix

		Predicted			
Emulated	Actual	CUBIC	Reno	Vegas	Veno
Low-$\widehat{diff}$	CUBIC	12	0	1	0
	Reno	0	10	0	0
	Vegas	0	0	9	0
	Veno	0	0	0	8
High-$\widehat{diff}$	CUBIC	10	0	1	0
	Reno	0	11	0	0
	Vegas	0	0	9	0
	Veno	0	1	0	8
Realistic					
Low-$\widehat{diff}$	CUBIC	13	0	1	0
	Reno	0	9	0	0
	Vegas	0	0	9	0
	Veno	0	0	0	8
High-$\widehat{diff}$	CUBIC	10	0	1	0
	Reno	0	10	0	0
	Vegas	0	0	9	0
	Veno	1	0	0	9



(a) Emulated Setting (b) Realistic Setting

Fig. 12: Empirical PDF estimations for the TCP protocols with a 95% confidence interval on emulated and realistic settings.

TABLE IV: Stochastic confusion matrix

	Emulated Setting					
	Predicted					
	Maximum difference			KS test		
Actual	Veno	Vegas	Cubic	Veno	Vegas	Cubic
Veno	99	0	1	100	0	0
Vegas	0	100	0	0	100	0
Cubic	2	0	98	5	0	95
Realistic Setting						
Veno	100	0	0	100	0	0
Vegas	0	100	0	0	100	0
Cubic	1	0	99	3	0	97

VI. Experimental Results and Discussion

We have conducted several experiments over different scenario settings. In order to justify and guarantee how our learning model could predict the development of a *cwnd* sawtooth and the underlying TCP variant with other realistic network traffic scenarios captured from the Internet, we created a realistic testbed as shown in Figure 4 where we experiment by running our resources on Google Cloud platform nodes deployed across the globe. As shown in Figures 9 and 10, our passive *cwnd* prediction model works reasonably well when applied both on an emulated and realistic evaluation scenarios. We confirm that our model operates correctly and accurately recognizes the sawtooth pattern for realistic scenario settings across different Google Cloud platforms. This shows that our prediction model is *general* bearing similarity to the concept of transfer learning in the machine learning community [29].

Evaluation metrics: The passive *cwnd* prediction was evaluated for accuracy using the Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE) metrics. The *cwnd* prediction performance result of both the emulated and realistic scenario settings across the Google Cloud platforms in terms of RMSE and MAPE is presented in Table V. As stated in Section IV, the ground truth data for the realistic setting was collected from the kernel of the TCP sending node. The performance results on both metrics indicate that our model is able to achieve reasonably accurate passive predictions of the development of *cwnd* sawtooth pattern.

TABLE V: cwnd prediction accuracy of *delay-based* TCP variants under an *emulated* and *realistic* settings

	Emulated Setting		*Realistic Setting*	
TCP Algorithms	**RMSE**	**MAPE (%)**	**RMSE**	**MAPE (%)**
Vegas [4]	1.8225	2.7618	3.6536	4.8864
Veno [11]	3.1421	3.8644	3.9254	4.8705
CUBIC [12]	4.0775	5.2961	3.6370	4.2774
Reno [17]	4.2484	5.9947	4.7541	5.0322

In our two-dimensional space analysis, we evaluated how the $\hat{\beta}$ varies as a function of the estimated queueing delay ($\widehat{diff}$) for all TCP protocols basing our hypothesis on the approaches presented on Section V. To see if the $\hat{\beta}$ is dependent on the queueing delay $\widehat{diff}$, let's consider TCP Veno [11]. Intuitively, If the value of $\widehat{diff}$ is *low* (i.e., $\widehat{diff}$<3), according to the standard specification Veno sets the β to 0.8 and it means Veno decreases the *cwnd* upon packet loss only by 20%. However,

if the delay is *high* (i.e., $\widehat{diff}$>3), Veno sets the β to 0.5. In case of TCP Vegas [4], if the $\widehat{diff}$ threshold is high enough Vegas increases the *cwnd* and when the *cwnd* doesn't reach the pipe, it decreases by 1. However, if the *cwnd* is pretty large, it converges the β towards 1 because of its Additive Increase and Additive Decrease (AIAD) strategy. In order to guarantee the accuracy of our TCP variant prediction model, we run our experiments where we ensure we have measurements with high $\widehat{diff}$ and low $\widehat{diff}$ values on different scenario settings. To this end, the prediction accuracies on emulated and realistic scenarios with these two measurement cases are 97.5%, 95%, 97.83%, and 95.46% respectively as shown in Table II and their corresponding confusion matrix is depicted in Table III. As explained above, our stochastic approach is also robust enough in terms of classifying the TCP variants based on the α parameter without the prior domain knowledge of the internal characteristics of each variant as it is shown in Figure 12.

VII. Conclusion and Future Work

In this paper, we investigate and explore in detail on how an intermediate node (e.g., a network operator) can identify the transmission state of delay-based TCP congestion control algorithms associated with a passively monitored TCP traffic. We present an effective TCP variant identification methodology from traffic measured passively by utilizing β, the multiplicative back-off factor to decrease the *cwnd* on a loss event, and the queueing delay values. We further employ a novel non-stationary time series approach from a stochastic nonparametric perspective using a two-sided Kolmogorov–Smirnov test to classify delay-based TCP algorithms based on the α, the rate at which a TCP sender's side *cwnd* grows per window of acknowledged packets, parameter. Our model is built in such a way that we don't want the sender or network change anything with the TCP parameter values that control increase and decrease ratios of the *cwnd*. Through extensive experiments on emulated and realistic scenarios, we have demonstrated that the data-driven classification techniques based on probabilistic models and Bayesian inference for optimal identification of the underlying delay-based TCP congestion control algorithms give promising and comparable results in terms of accuracy. In conclusion, we show that the learned prediction model performs reasonably well by leveraging trained knowledge from the emulated network when it is applied and transferred on a realistic scenario setting. Finally, we have shown that our model can also be applied equally well to loss-based TCP variants using the presented approaches. To the best of our knowledge, this paper is the first to study how the variability of the β parameter as a function of queueing delay and the α parameter can be used for passive TCP variant identification in real-time.

As part of our future work, we would like to substantially extend this work in terms of devising a generic learning model for operating system fingerprinting from passive measurements by combining the basic TCP/IP features and the underlying TCP variant as input vectors.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, 2016.

[2] A. A. Abouzeid and S. Roy. Stochastic modeling of TCP in networks with abrupt delay variations. 2003.

[3] E. Altman, K. Avrachenkov, and C. Barakat. A stochastic model of TCP/IP with stationary random losses. *ACM SIGCOMM Computer Communication Review*, 30(4):231–242, 2000.

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

[5] D.-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 1989.

[6] T. M. Cover, P. E. Hart, et al. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 1967.

[7] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester. *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer, 2005.

[8] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[9] ESnet. iperf3. https://iperf.fr/iperf-servers.php, 2017.

[10] G. Fasano and A. Franceschini. A multidimensional version of the Kolmogorov–Smirnov test. 1987.

[11] C. P. Fu and S. C. Liew. TCP Veno: TCP enhancement for transmission over wireless access networks. *IEEE*, 2003.

[12] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS*, 2008.

[13] D. H. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure. General TCP State Inference Model from Passive Measurements Using Machine Learning Techniques. *IEEE Access*, 2018.

[14] D. H. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure. Recurrent Neural Network-Based Prediction of TCP Transmission States from Passive Measurements. In *NCA*. IEEE, 2018.

[15] D. H. Hagos, P. E. Engelstad, A. Yazidi, and O. Kure. Towards a Robust and Scalable TCP Flavors Prediction Model from Passive Traffic. In *ICCCN*. IEEE, 2018.

[16] S. Hemminger et al. Network emulation with NetEm. 2005.

[17] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM computer communication review*. ACM, 1988.

[18] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM Computer Communication Review*, 1989.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] S. Kullback. *Information theory and statistics*. 1997.

[21] D. J. Leith, R. N. Shorten, G. McCullagh, L. Dunn, and F. Baker. Making available base-RTT for use in congestion control applications. *IEEE Communications Letters*, 2008.

[22] J. Oshio, S. Ata, and I. Oka. Identification of different TCP versions based on cluster analysis. In *ICCCN 2009*. IEEE, 2009.

[23] S. Ostermann. Tcptrace. http://www.tcptrace.org, 2000.

[24] M. Panda, H. L. Vu, M. Mandjes, and S. R. Pokhrel. Performance analysis of TCP NewReno over a cellular last-mile: Buffer and channel losses. *IEEE Transactions*, 2015.

[25] J. Postel et al. Transmission control protocol. RFC 793, 1981.

[26] H. Tang and J. Glass. On Training Recurrent Networks with Truncated Backpropagation Through Time in Speech Recognition. *arXiv preprint arXiv:1807.03396*, 2018.

[27] Z. Wang and J. Crowcroft. A new congestion control scheme: Slow start and search (Tri-S). *ACM SIGCOMM*, 1991.

[28] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM transactions on Networking*, 2006.

[29] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big Data*, 2016.

[30] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *INFOCOM*. IEEE, 2004.

[31] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu. TCP congestion avoidance algorithm identification. *IEEE*, 2014.