Forecasting TCP's Bandwidth to Speed Up Slow Start

Ralf Lübben

Selection of the optimal transmission rate in packet-switched best-effort networks is challenging. Typically, senders do not have any information about the end-to-end path and should not congest the connection but at once fully utilize it. The accomplishment of these goals lead to congestion control protocols such as TCP Reno, TCP Cubic, or TCP BBR that adapt the sending rate according to extensive measurements of the path characteristics by monitoring packets and related acknowledgments. To improve and speed up this adaptation, we propose and evaluate a machine learning approach for the prediction of sending rates from measurements of metrics provided by the TCP stack. For the prediction a neural network is trained and evaluated. The prediction is implemented in the TCP stack to speed up TCP slow start. For a customizable and performant implementation the extended Berkeley packet filter is used to extract relevant data from the kernel space TCP stack, to forward the monitoring data to a user space bandwidth prediction, and to feed the prediction result back to the stack. Results from a online experiment show improvement in flow completion time of up to 30%.

***Index Terms*—artificial intelligence (AI), computer network performance, machine learning (ML), performance forecast, transmission control protocol (TCP), slow start**

I. Introduction

In packet switch networks that are based on the best effort principle, the selection of an optimal sending rate is challenging. Typically, the sender has no a priori knowledge of the path and has to determine the rate empirically with the goal to utilize the path fully without overloading it. This problem is typically solved by congestion control protocols such as TCP Reno, TCP Cubic, or TCP BBR. TCP Reno and Cubic increase the datarate iteratively until a packet loss occurs. This loss is interpreted as a signal for congestion and the sending rate is reduced. This means, that these algorithms produce short-term overload to cause a loss. Nevertheless, packet loss do not only occur by overload of a path but may also be caused by corruption of data packets, e.g., from interference in wireless networks. Such losses are interpreted as congestion, too, and thereby lead to unnecessary adaptations of the rate. TCP Reno and TCP Cubic differ in the way the sending rate is adapted, in detail the value of the congestion window, i.e., the number of packets, that can be sent in a round-trip time (RTT), is adapted differently. TCP BBR follows a different approach, it measures the bandwidth and RTT to determine the optimal number of packets to be sent in a round-trip time, therefore it avoids the short-term overload. In detail, it also starts with a startup phase, i.e., the slow-start algorithm, afterwards it enters into the drain phase to empty a possibly self-induced queue, and then changes iteratively between two phases to probe the bandwidth and the RTT. Details of the phases and evaluation are given in [1], [2], [3]. All protocols have in common, that in the beginning they perform a slow-start to determine an operation point fast.

These algorithms derive the next state, i.e., the congestion window size or sending rate from recent measurements of the path. We propose an in-band and passive throughput prediction to forecast the expected throughput, which is attainable in the near future. By in-band and passive, we mean a method that uses information already available at transport protocols without the need to modify the TCP protocol. In detail, the bandwidth measurement performed by TCP BBR is used to train a neural network. As we elaborate in Sec. III-A, we only require an extension to monitor internals of the TCP stack, which can rely on recent extensions on the Linux operating system. In detail, we look at the slow start phase, in which TCP probes for the attainable throughput. We envision to speed up the rate adaptation during the start-up phase from which especially flows with a short duration benefit from. Such flows occur frequently in real-world networks [4].

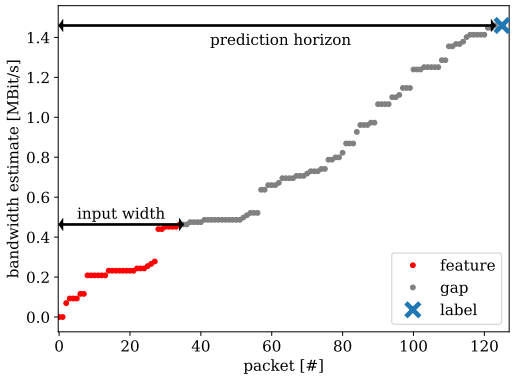


Fig. 1. BBR rate estimates captured from the network stack, the blue markers present amongst others the input features to predict the future state presented by the green marker, so this prediction can potentially use to skip the iterative adaptation shown as red markers.

Fig. 1 shows an exemplary trace of the rate estimate performed by TCP BBR as recorded by the TCP stack. We present details of the internal variables in Sec. III, here, we limit the description to the basic idea. Our forecast predicts the state of the TCP stack by observation of the beginning of the connection (red markers), i.e., we observe the statistics of the first packets of the connection, amongst others the rate estimate, as example of multiple features, to predict the future state. This state is shown as blue marker. In this article, we first contribute on forecasting TCP's performance by a neural network and by the use of extensive monitoring of the network stack. We evaluate the prediction offline in terms of

Ralf Lübben is with the Flensburg University of Applied Sciences, Kanzleistrasse 91-93, Flensburg, Germany (e-mail: ralf.luebben@hs-flensburg.de).

the selection of input features, the input width, the prediction horizon, and perform a hyperparameter optimization.

From this supervised learning, we integrate the trained neural network into the TCP stack to forecast the attainable rate from the observation of the beginning of the TCP connection and evaluate the performance improvement achieved by the prediction. Here, we make use of the extended Berkeley Packet Filter (eBPF), a virtual machine inside the Linux, which gives us a flexible approach to extend TCP and apply a neural network for prediction as a user space program using a basic machine learning software framework.

The structure is as follows: Sec. II presents the related work with a focus on a background on information that can be inferred from observations of packets. Sec. III specifies the details of the neural network and the parameters we evaluate. The performance of the forecast is given in Sec. IV by an offline and online analysis. A conclusion and outlook is given in Sec. V.

II. Related Work

For the related work, we look into the field of congestion control and bandwidth estimation first, i.e., how capabilities can be determined and protocols can adapt to it. Second, we summarize related work on the application of machine learning to this field.

Usually, in packet switched networks end hosts do not have any information about the end-to-end link capabilities in terms of available bandwidth and latency. Moreover, these values are time-varying due to multiplexing of traffic flows at various intermediary devices and links with time-varying performance, such as wireless networks in which capacity can change rapidly. End-hosts have to adapt to these time-varying capabilities, so that the link is not overloaded but the connection utilizes the capacity it is entitled for. Common congestion control algorithms strive at a fair distribution between multiple flows at a bottleneck. The congestion control algorithms TCP Reno, defined in [5], and TCP Cubic, described in [6], implement it by an iterative adaptation of the congestion window based on packet loss events. The congestion control protocol determines the amount of data a sender transmit in one round trip time, therefore it reflects the sending rate. The iterative adaptation, i.e., an increase of the window until a loss occurs and the successive reduction, leads to a continues change between overload and underload of the path. TCP BBR avoids this iterative change by the use of an estimation method of the attainable rate of an end-to-end connection. The basic design follows approaches from the field of bandwidth estimation, whereas TCP BBR uses passive measurements, i.e., existing data packets are used for probing the network, see [1].

In the field of bandwidth estimation, as defined in [7], [8] as the unused capacity of a path, often active measurements are used to infer the network capabilities from measurements, since passive measurement exhibit some drawbacks, see [9]. The use of active measurements allows for a better control of the arrival process to infer more information from the network path. Examples are packet trains, packet pairs, or packet chirps. Packet chirps are a train of packets, whereas the gap between the packets is decreasing and thereby the rate is increasing. Therefore, the chirp probes the network with various data rates using only a short train. Usual packet trains are sent at fixed rate. Related examples are e.g. the tool Pathload, which uses packet trains [10], Pathchirp [11], which introduces chirps, and Spruce [12], that measures gaps between packets. The available bandwidth expresses the connection by one number and thereby neglects the time-varying characteristic. More complete descriptions are service curves from the framework network calculus, which provides descriptions for deterministic as well as stochastic networks [13]. Techniques for the estimation of service curves are described in [14], [15], [16].

Still, bandwidth estimation often relies on assumptions of a system model including specific scheduling disciplines and does not encounter effects from operating systems such as memory limitations. Common pitfalls in bandwidth estimation are given in [17]. Therefore, we propose in the next section to utilize machine learning to infer the rate a TCP connection can attain. On the one hand, from the previous work, basic relations are known, on the other hand, correlations at least on short timescales in path characteristics are known, see e.g. [18], [19], so that we expect that future states can be derived from measurements and leave the process of refinement to the machine learning approach. By the collection of extensive measurements of the TCP stack, our approach also comprises details such as memory constraints from operating systems. Before presenting our approach in the next section, we summarize work done in the application of machine learning to congestion control and bandwidth estimation before. For a comprehensive summary, we refer to the survey [20].

The application of machine learning to congestion control is manifold, it is used for prediction of throughput, RTTs, the congestion window, or even to learn the congestion control algorithm itself. Measurements are collected during the connection is active or also gathered by active or passive measurements before the actual connection starts. Also, the training may occur supervised on historical data or unsupervised during the connection is active, typically by optimization of a utility function. Machine learning applied to the estimation of available bandwidth estimation is explored in [23], [24], [25]. The authors show the higher accuracy achieved by supervised machine learning, which leads to a reduction of probing traffic if compared to conventional estimation tools. The applicability in the context of TCP is evaluated in [26], [25]. In these works no active probing traffic is required, instead the timings between TCP packets are exploited. In [26], the improvement of reinforcement learning to estimate the available bandwidth is shown. Typically, the estimate is relevant for rate adaptation or congestion control, for which we introduce the related work next.

In [27], the bulk transfer throughput of TCP is predicted by support vector regression, which shows that machine learning can outperform prediction methods that rely on mathematical models or infer the throughput from the packet trace history. For practical measurements, the authors rely on a packet train probing before the actual TCP connection to predict the bulk transfer throughput.

In [28], the end-to-end transport protocol Sprout is de-

TABLE I
INPUT FEATURES EXTRACTED FROM THE LINUX TCP NETWORK STACK

parameter	description	modification
ts	timestamp	increment
ca_state	congestion state	
total_retrans	total number retransmissions	increment
probes	number of probes sent	
rto	retransmission timeout	
last_data_sent	time interval last data sent	
last_data_recv	time interval last data received	
last_ack_recv	time interval last ack received	
rtt	current rtt estimate	
rttvar	current rtt variance estimate	
snd_cwnd	congestion window	
pacing_rate	pacing rate	
bytes_acked	bytes which are acked	increment
notsent_bytes	bytes not sent	
min_rtt	current minimal RTT estimate	
delivery_rate	delivery estimate	
busy_time	cumulative busy time of the connection	increment
sndbuf_limited	cumulative time the flow was limited by the send buffer	increment
delivery_rate_app_limited	cumulative time the flow was limited by no application data available	increment
bytes_retrans	bytes retransmitted	
reord_seen	number of reordering events	
bbr_bw	bandwidth estimate of TCP BBR	
bbr_minrtt	minimal RTT determined by TCP BBR	
dsack_dups	report of duplicate SACK segements [21], [22]	
rwnd_limited	cumulative time the flow was limited by the receive buffer	increment
delivered	data packets delivered incl. retransmissions	increment
bytes_sent	number of octets of data contained in transmitted segments, including retransmitted data [22]	increment
segs_out	number of segments sent containing a positive length data segment	increment

veloped, which uses a packet delivery forecast based on a Poisson process with variable rate. This rate parameter is adapted to the path capabilities to determine a sending rate, so that the probability that packets get queued is small. A congestion control algorithm designed by an automatic learning process from timestamps of sent and received packets and RTT estimates is presented in [29]. The results show that algorithms that are trained automatically outperform designed algorithms such as TCP Reno or TCP Cubic. A series of approaches that learn the actions based on observation of the current network performance to maximize a utility function are presented in [30], [31], [32]. Therefore, the congestion control algorithm is designed during the connection. Approaches that apply reinforcement learning to decide on actions for congestion control are presented in [33], [34] The approach presented in [35] implements machine learning to predict the next congestion window value after a congestion event such as packet loss. It uses multiple values measured by the TCP stack, the results show that using a large set of information from the stack improves the prediction.

The prediction of the RTT as replacement for the exponential weighted average RTT estimator using online learning is presented in [36]

In the scope of the related work, we rely on the bandwidth estimation implemented in TCP as training data, which estimates the delivered packets in a time interval. The estimation method is described in [37]. Therefore, it refers to a packet train estimation method, which are common in bandwidth estimation. Furthermore, we rely on this estimate to train a neural network from multiple values determined by the TCP stack to predict this value. By the use of the existing estimation technique, we are able to collect training data practically without extension of TCP, except a monitoring hook, or integration of additional measurements in the network. The related neural network is afterwards used to assist TCP BBR to select a bandwidth during slow start. Therefore, the proposed solution assists TCP BBR but does not generate new congestion control protocols in comparison to the proposals in [30], [31], [32].

III. DATA GENERATION AND PREDICTION

We present in this section, on the hand, the model we use for the prediction and, on the other hand, describe the experiments and metrics we gather for training the model.

A. Data Generation

The data we use for training the neural network originates from a network experiment. To gather samples from a lot of divers flows, we create an emulated network experiment as illustrated in Fig. 2. The network consists of a load generator client that downloads a file of size 3.2 MB via HTTP from the server. This client generates 1000 requests per minute in average. To schedule the requests the load testing software JMeter is used. It schedules requests by an Poisson arrival process. With a segment size of 1440 bytes and 60 bytes headers, i.e., a flow transmits about 2330 packets to download the file with an average of 17 flows per second, the average rate is about 425 Mbit/s. Each experiment is executed for 120 sections and repeated 5 times.

We perform various experiments to achieve a divers data set. For the experiment described above, we vary the network parameters. In detail, we set the one way delay to one of 25, 50, 75, 100, 125, 150, 175, 200 ms with a jitter of a twentieth and configure the buffer size to the bandwidth delay product

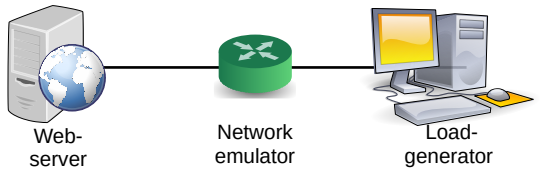


Fig. 2. Experiment setup: The load generator downloads simultaneously the requested file from the web server to measure the flow completion time and collect TCP statistics in various network setups.

(BDP) and half of the BDP. In the end, we extract 150,000 flows from these experiments. These samples are split into a train, validation, and test data set. The validation and test data set contain each 15,000 of these samples. All evaluation results presented later regarding the neural network use the test data set, whereof the samples are not used during training.

To collect the internal TCP statistics from the Linux TCP BBR networking stack at the server, we create an eBPF program, i.e., a program that is executed inside the Linux kernel. Thereby, we record extensive information of the TCP stack given in the struct `tcp_info` for each packet. The values used for training are given in Tab. I. We note here, that the struct includes more values, which are recorded, too, but not included for the training, since they are constant during measurements or include irrelevant data such as port numbers. Nevertheless, these values may be useful in other scenarios. One example is the client's IP address. In our experiment it is constant, in real-world networks it may be useful for training, since the client's network may impact the performance. Some parameters are cumulative and increasing sequences, such as timestamps (ts), cumulative times (e.g. busy_time), byte or packet counters (e.g. bytes_acked). For these values, we calculate the increments of the sequence and use the increments for the training. The timestamp `ts` is typically derived from the uptime of the system, so we compute the increments and shift the sequence that each trace starts at $ts = 0$. This returns the interarrival times of the packets.

The building blocks for gathering of TCP statistics and the components for the prediction, which is described in Sec. IV-C, are presented in Fig. 3. The implementation as eBPF program gives a customizable solution for the monitoring and prediction. Since Linux kernel version 5.13, selected congestion control functions of the TCP stack can easily be extended. It adapts to the data required for the prediction, whereas the implementation of the prediction can be implemented in the user space and offers an extensible solution to implement and adapt prediction methods in common machine learning frameworks.

B. Bandwith Prediction - Problem Formulation

We model the prediction as a generic regression problem with

$$y(n+g) = f(X(0,n), \beta) + e \tag{1}$$

whereas $y(n+g)$ is the rate estimate measured for packet $n+g$ by the TCP network stack, $X(0,n)$ the various input features observed from the TCP stack for the packets 0 to n, β the

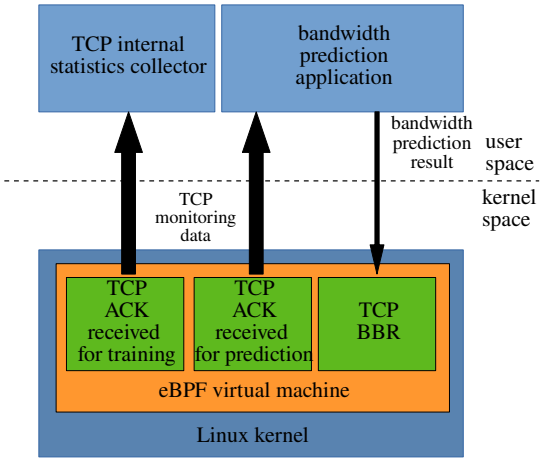


Fig. 3. Building blocks of the deployment using eBPF: TCP internal measurements are transferred to user space applications to collect data or for bandwidth prediction.

model parameters, and e the error term, which is not directly observable, this may include the time-varying behavior of links in the network or the cross-traffic multiplexed in the network.

The assumption is that a future value of the delivery rate is a good estimate of the attainable rate of the connection. This assumption is based on basic relations between timings of packets and data rates, such models are the Lindley equation, relation of inputs and outputs of packets and variations of packet trains, known from bandwidth estimation as elaborated in Sec. II.

We use here the term attainable rate, since the available bandwidth is defined as the unused capacity in a time interval [7], [8]. The attainable rate may defer in dependence of the scheduling discipline. For example using TCP and a FIFO scheduling at the bottleneck link, TCP targets at a fair share, i.e., if the available bandwidth is below the fair share, the attainable rate exceeds the available bandwidth. Still, TCP does not always achieve the fair share [3], [38]. Here, we rely on the estimate of TCP BBR and do not account for shortcomings of the protocol itself.

In detail, we focus here on the start-up phase of the connection, due to two reasons. At the beginning TCP sends a burst of packets, at first ten packets, which are doubled each RTT until a loss occurs. On the basis of bandwidth estimation such packets represent a packet train, which provides a reliable estimate, see e.g. [10]. We make use of this estimation implemented in the TCP stack to obtain our training data, i.e., we monitor the estimate for packet $n+g$ and extract the additional features for packets zero to n $X(0,n)$ the stack provides, see. Tab. II.

We further assume this estimate contains information on the future network performance. This implies a correlation and causality between the observations and the later value, this is given by the congestion control algorithm and the system model, whereas random hidden events such as cross-traffic, variability of physical links, especially for wireless connections, and loss events may occur. Furthermore, related works show that stationarity at least for short-time scales is

TABLE II
INPUT FEATURES

feature	values
Input width	5, 15, 35
Packet number	62, 125, 250, 500, 1000, 2000

often observable [18], [19] in computer networks.

Still, the models for bandwidth estimation mentioned in Sec. II base the foundation on specific assumptions, which may not always be met in practice. Many other effects such as memory limitations at the sender or receiver, different scheduling disciplines, random cross-traffic, and time-varying link characteristics influence such theoretical models [39], which a neural network can incorporate if trained in a specific setup.

From these causalities between the observations of information from the TCP stack, stationarity in networks, relations between available bandwidth and packet timings, we expect a machine learning approach can infer the attainable rate of a connection from observation of the start-up phase of the connection. We evaluate our hypothesis in Sec. IV.

C. Neural Network Design

To create a model for Eq. (1), we rely on a Long-Short-Term-Memory (LSTM) network, which is known to model the probability of the next input by the previous outputs as described in [40]. Thereby the network learns also from the history of previous inputs and outputs. The schema reflects the relationship known for input/output relations of packets in computer networks, since outputs depend on previous inputs as modeled by the Lindley's recursion [41], in framework of the network calculus [13], or characteristics of network traffic such as self-similarity [42]. The prediction is also tested with a deep neural network with up to three hidden layer that shows a similar performance in this setup.

For the training, we perform a hyperparameter optimization and a feature selection. The features, we evaluate, are given in Tab. II and the hyperparameters, we test, are given in Tab. III. We limit the input width up to 2000 packets, since the overall connection transfers about 2300 packets as described in Sec. III-A. The input width is adjusted to the TCP slow start behavior. Usually, when the connection starts the TCP sender transmits in the first round of the slow start 10 packets, when acknowledgments arrive, in the next round 20 packets, and afterwards 40 packets until a loss occurs. For example, in the first round, when 10 packets are sent out, the sender has to wait about a round trip time to receive acknowledgments. Therefore we select input widths that fall into each of these rounds. For the proposal in Sec. IV-C, this means after these number of packets the bandwidth prediction is triggered, so that the prediction result can immediately apply the corresponding round. This is demonstrated in Fig. 4, the green dot indicates an input width of five acknowledgments for which the prediction is triggered. An inappropriate choice is marked by the red dot as end of the round of returning acknowledgments. Following packets will only be transferred after new acknowledgments have been received. The packet number for which the prediction is performed is arbitrarily chosen in a nearly exponential increasing order. For the adaptive learning rate, we reduce from 0.01, 0.001, 0.0001, and 0.00001 at each quarter of epochs.

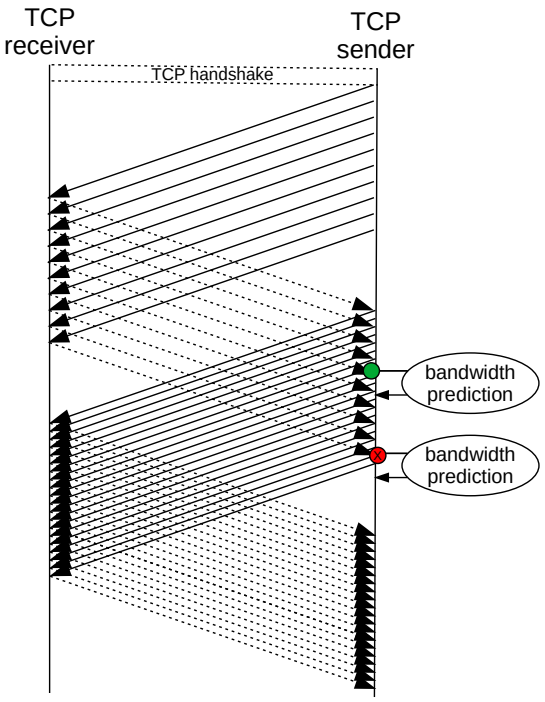


Fig. 4. TCP slow start behavior and optimal (green dot) and non-optimal (red dot) points to trigger the bandwidth prediction.

TABLE III
PARAMETERS FOR HYPERPARAMETER OPTIMIZATION.

parameter	values
learning rate	adaptive
LSTM cells	5, 10, 20, 40, 70, 150, 300, 600
l2 kernel regularizer	0.0, 0.01,
l2 recurrent regularizer l2	0.0, 0.01
l2 bias regularizer	0.01, 0.001
optimizer	adam
dropout	0.0, 0.1, 0.2
epochs	300

For hyperparameter optimization, we use random search as implemented in keras tuner [43] with 50 trials and optimize for the mean absolute percentage error:

$$l_{MAPE} = \left| \frac{y - \hat{y}}{y} \right|,$$

where $\hat{y}$ is the predicted value.

IV. RESULTS

We start the result section with the analysis of the data set to identify the bandwidth estimated by the TCP stack after a given number of packets transmitted, see Tab. II, and furthermore describe the quality of experience by the evaluation of the flow completion time (FCT). Afterwards, we provide results of an intensive training of the bandwidth by the use of the data set. Finally, we apply the best training result

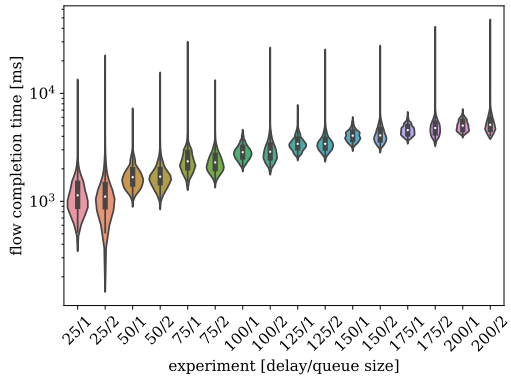


Fig. 5. FCT in dependence of delay and queue size. The x-label defines the delay as well as the BDP, a 1 means it equal the BDP, a 2 means it equals halve the BDP.

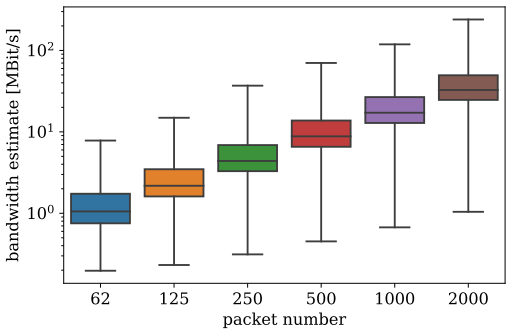


Fig. 6. Bandwidth as estimated by the TCP stack for a specific packet in the connection. Each box shows the meadian, 25% and 75% percentiles, and minimal and maximal value. The bandwidth estimate increases with the packet number, since it can at most reflect the sending rate, at the beginning if a flow it is typically small and increases due to TCP slow start.

in an online analysis, in which the prediction is implemented in the TCP stack.

A. Data Description

For the data analysis, we evaluate as quality of experience metric the FCT for the different experiment's configuration in terms of delay and queue size, this is also the metric, we finally evaluate for effectiveness of our proposal. The second metric is a quality of service metric, in detail the bandwidth estimate of the TCP stack, we predict. Fig. 5 presents the FCT grouped by the individual network configurations of delay and queue size, i.e. a factor of one means the queue size equals the BDP, a factor of two means the BDP is half the BDP. As expected, the FCT increases with increasing delay, the median value ranges from about one second to about five seconds. Especially with a smaller queue size, half of the BDP, the FCT shows some large outliers due to packet loss at the intermediate router. This result is also our reference for later comparison in the online experiment in Sec. IV-C.

For the prediction, the bandwidth estimate of the TCP stack is used. Fig. 6 shows the median, 25%, 75% percentile and min and max values of the bandwidth estimate performed by the TCP stack for a specific packet number. The estimate increases with the packet number observed. This follows from the probing mechanism used by TCP, since it uses a packet train, the estimate can reflect at most the sending rate, which is small at the beginning of the connection and increases due to slow start and is adapted to congestion avoidance in TCP. In this experiment, we use the estimate of a specific packet, since flows are short and typically the slow start phase is not quitted. In a different scenario, the estimate when the slow start phase is left and congestion avoidance takes over, may reflect a better estimate of a connection.

To identify the performance improvement, we put into relation the bandwidth estimate as measured by the TCP stack for the last packet of the input widths, see Tab. II. Fig. 7 displays this relation, the highest improvement is achieved for a small input sequence of five packets to the estimate of packet 2000. Therefore, a estimate with a short input width and a

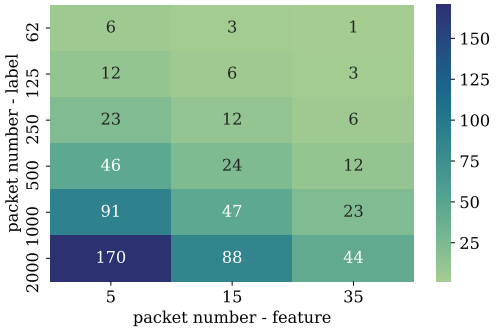


Fig. 7. Relation of the bandwidth estimate of packet {62, 125, 250, 500, 1000, 2000} to estimate related to the last packet of the input sequence. The bandwidth estimate of a small input width and a large packet number gives the highest improvement.

prediction of the bandwidth of packet 2000 is desirable. We evaluate, the prediction in the next section.

B. Offline Analysis

We train the neural network to predict the bandwidth estimate for a prediction horizon of packets {62, 125, 250, 500, 1000, 2000} using different input widths as listed in Tab. II. The results with the lowest mean absolute percentage error are displayed in Fig. 8. Obviously, the prediction shows the smallest error if the prediction horizon is short, i.e. a large input width and a packet in the near future. Still, the expected gain is expected to be small if compared to Fig. 7. For a large prediction horizon, the error increase to about 20%, even for a small input width and a large horizon the error stays at about 24%.

This first evaluation contains all listed features. We further conduct an evaluation of reduced feature sets. For the reduced feature selection, we perform two basic evaluations. For one training, we use a minimal feature set, i.e., the features *ts* and *bbr_bw*, and for a second training additionally the *RTT*. These experiments give information about the benefit of the additional parameters besides these basic parameters. These features are selected due to basic analytical relationships in

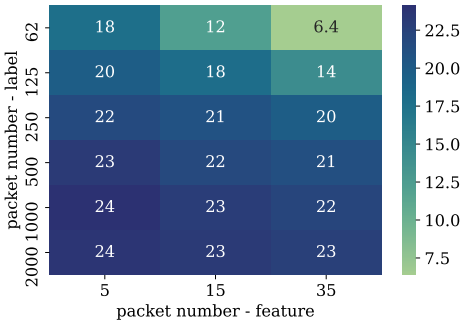
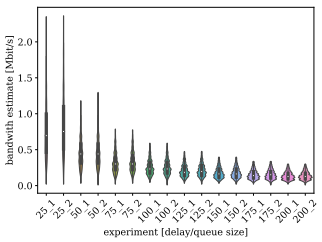
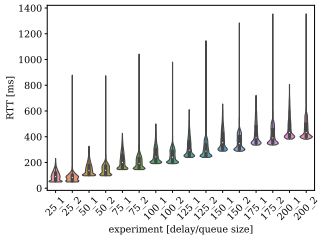


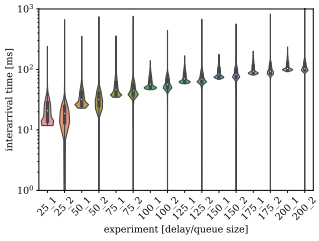
Fig. 8. Mean absolute percentage error of the prediction on the test data set, on long input sequences and small packet numbers the prediction performs best, still for small input widths and high packet numbers the error is about 25%.



(a) bbr_bw



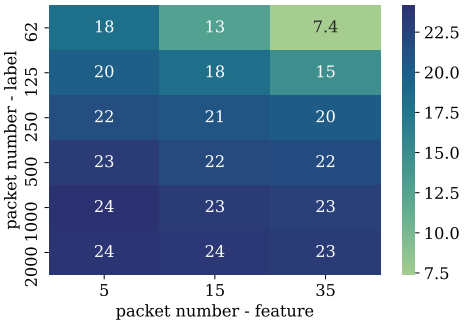
(b) RTT



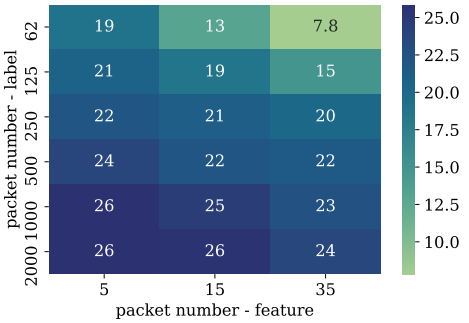
(c) ts

Fig. 9. Distributions of selected features displayed as violin plot: Between different delay configuration the ranges are distinguishable from each other, still their ranges overlap.

computer networks. Common models in the area of queueing theory rely on timestamps. For example, the Lindley equation calculates waiting times in queueing networks from interarrival time of packets and service times. In the framework of network calculus the concept is generalized, whereat the departure times are deduced from the arrival process, i.e. arrival times of packets to a network, and a service process. Furthermore, the importance of the RTT follows from the basic principle of TCP, which is self-clocked due the arrivals of acknowledgements and the window mechanism, that decides on the amount of data to be sent in one RTT. For TCP BBR the congestion window is derived from the bandwidth estimate performed by TCP, therefore we include this value *bbr_bw*, too. Fig. 9 displays the distributions of these features for the different experiments. Between experiments using a different delay configuration the distributions differentiate clearly, but also overlap in their range, especially between experiments that have a configured delay, which is close to their own configured



(a) ts, bbr_bw, rtt



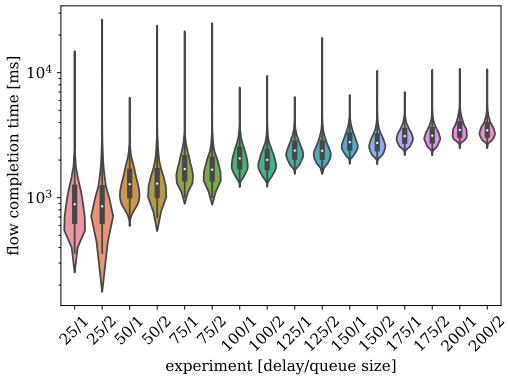
(b) ts, bbr_bw

Fig. 10. MAPE for minimal feature sets, compared to Fig. 8 the MAPE increases especially for large input width and small packet numbers, for a large prediction horizon and small inputs width the MAPE is similar on the minimal feature sets.

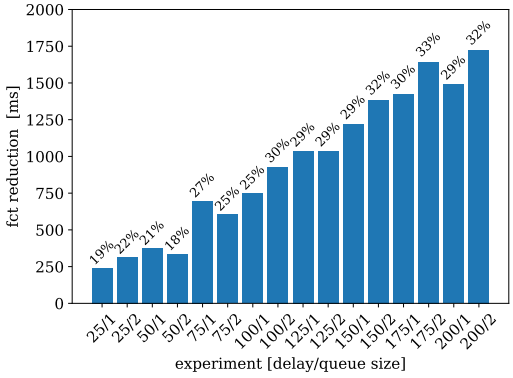
delay. Therefore, these values are eligible for a prediction, but still are challenging due to a overlap in their value. Fig. 10 presents the loss for the minimal feature set. Already these basic feature give similar results as the full feature set evaluated previously and depicted in Fig. 8. Especially, on short prediction horizons the full feature set performs better, on a small input width of five packets and a large horizon to packet number 2000 the set of *ts*, *bbr_bw*, *RTT* performs equal to the full feature set, leaving out the RTT values increases the error only by two percent.

C. Online Analysis

As presented in Fig. 3, we implement a bandwidth prediction and integrate it into the TCP stack by the use of eBPF. We use a trained model from Sec. IV-B, in detail, the model for a input width of five packets and estimation of the bandwidth for packet 2000, which gives the largest improvement. We repeat the experiment described in Sec. III-A and measure again the FCT for the different network scenarios. Each setup is repeated 30 times. The FCT and the reduction in delay compared to the former experiment are shown in Fig. 11. For all scenarios the FCT decreased, whereby the relative reduction increases for scenarios with a larger RTT, since the RTT has a strong impact on the performance. The reduction ranges from about 20% to more than 30%.



(a) ts, bbr_bw



(b) all features

Fig. 11. FCT of the network scenarios using a bandwidth prediction and related reduction in FCT in percent compared to the unmodified TCP behavior.

V. Conclusions

In this article, we describe the prediction of the attainable rate of TCP connections. We present the impact of different features by an offline analysis such as the length of the observation, the length of the prediction horizon, and selected features from extensive monitoring of variables available in the TCP stack on the performance of a neural network for the prediction. This neural network is implemented in an online variant, to predict the attainable rate and usage of the estimate to speed up the slow start phase of TCP. The results shown an improvement from 20% to more than 30% in dependence of RTT. In future work, we plan to integrate the approach into real-world networks to collect addition training data and evaluate in the performance in the Internet.

References

[1] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," vol. 14, no. 5, 2016.

[2] "Reproducible measurements of tcp bbr congestion control," *Computer Communications*, vol. 144, pp. 31–43, 2019.

[3] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of tcp bbr congestion control," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, 2018, pp. 1–9.

[4] S. Bauer, B. Jaeger, F. Helfert, P. Barias, and G. Carle, "On the evolution of internet flow characteristics," in *Proceedings of the Applied Networking Research Workshop*, ser. ANRW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 29–35.

[5] RFC 6582.

[6] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," vol. 42, no. 5, 2008.

[7] X. Liu, K. Ravindran, and D. Loguinov, "A queuing-theoretic foundation of available bandwidth estimation: Single-hop analysis," vol. 15, no. 4, pp. 918–931, Aug. 2007.

[8] ——, "A stochastic foundation of available bandwidth estimation: Multi-hop analysis," vol. 16, no. 1, pp. 130–143, Apr. 2008.

[9] J. Liebeherr, M. Fidler, and S. Valaee, "A system theoretic approach to bandwidth estimation," vol. 18, no. 4, pp. 1040–1053, Aug. 2010.

[10] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 537–549, 2003.

[11] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "PathChirp: Efficient available bandwidth estimation for network paths," in *Proc. PAM*, Apr. 2003.

[12] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proc. ACM IMC*, Oct. 2003, pp. 39–44.

[13] M. Fidler, "A survey of deterministic and stochastic service curve models in the network calculus," vol. 12, no. 1, pp. 59–86, Feb. 2010.

[14] R. Lübben and M. Fidler, "Service curve estimation-based characterization and evaluation of closed-loop flow control," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 1, March 2017.

[15] R. Lübben, M. Fidler, and J. Liebeherr, "Stochastic bandwidth estimation in networks with random service," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 484–497, April 2014.

[16] N. Becker and M. Fidler, "A non-stationary service curve model for estimation of cellular sleep scheduling," vol. 18, no. 1, 2019.

[17] M. Jain and C. Dovrolis, "Ten fallacies and pitfalls on end-to-end available bandwidth estimation," in *Proc. ACM IMC*, Oct. 2004, pp. 272–277.

[18] G. Urvoy-Keller, "On the stationarity of tcp bulk data transfers," in *Passive and Active Network Measurement*, C. Dovrolis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 27–40.

[19] Y. Zhang and N. Duffield, "On the constancy of internet path properties." New York, NY, USA: Association for Computing Machinery, 2001.

[20] "When machine learning meets congestion control: A survey and comparison," *Computer Networks*, vol. 192, p. 108033, 2021.

[21] RFC 2883.

[22] RFC 4898.

[23] "Online available bandwidth estimation using multiclass supervised learning techniques," *Computer Communications*, vol. 170, pp. 177–189, 2021.

[24] "Machine learning for measurement-based bandwidth estimation," *Computer Communications*, vol. 144, pp. 18–30, 2019.

[25] S. K. Khangura, "Neural network-based available bandwidth estimation from tcp sender-side measurements," in *2019 8th International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, 2019, pp. 1–6.

[26] S. K. Khangura and S. Akın, "Measurement-based online available bandwidth estimation employing reinforcement learning," in *2019 31st International Teletraffic Congress (ITC 31)*, 2019, pp. 95–103.

[27] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A machine learning approach to tcp throughput prediction," *IEEE/ACM Transactions on Networking*, vol. 18, no. 4, pp. 1026–1039, 2010.

[28] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. USA: USENIX Association, 2013, p. 459–472.

[29] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control." New York, NY, USA: Association for Computing Machinery, 2013.

[30] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 395–408. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong

[31] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC vivace: Online-learning congestion control," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 343–356. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/dong

[32] T. Meng, N. R. Schiff, P. B. Godfrey, and M. Schapira, "PCC Proteus: Scavenger Transport And Beyond." New York, NY, USA: Association for Computing Machinery, 2020.

[33] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 3050–3059.

[34] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "Qtcp: Adaptive congestion control with reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 445–458, 2019.

[35] L. Bai, H. Abe, and C. Lee, "Rnn-based approach to tcp throughput prediction," in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, 2020, pp. 391–395.

[36] B. A. A. Nunes, K. Veenstra, W. Ballenthin, S. Lukin, and K. Obraczka, "A machine learning approach to end-to-end rtt estimation and its application to tcp," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011.

[37] Y. Cheng, N. Cardwell, S. H. Yeganeh, and V. Jacobson, "Delivery Rate Estimation," Internet Engineering Task Force, Internet-Draft draft-cheng-iccrg-delivery-rate-estimation-01, Nov. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-cheng-iccrg-delivery-rate-estimation-01

[38] T. Lakshman and U. Madhow, "The performance of tcp/ip for networks with high bandwidth-delay products and random loss," *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, pp. 336–350, 1997.

[39] M. Bredel and M. Fidler, "A measurement study of bandwidth estimation in ieee 802.11g wireless lans using the dcf," in *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, A. Das, H. K. Pung, F. B. S. Lee, and L. W. C. Wong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 314–325.

[40] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

[41] "The theory of queues with a single server," vol. 48.

[42] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, "Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 71–86, 1997.

[43] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, "Keras Tuner," https://github.com/keras-team/keras-tuner, 2019.