Evaluating TCP Throughput Predictability from Packet Traces using Recurrent Neural Network

Ryu Kazama
University of Tsukuba, Japan

Hirotake Abe
University of Tsukuba, Japan

Chunghan Lee
Toyota Motor Corporation, Japan

2022 IEEE Symposium on Computers and Communications (ISCC) | 978-1-6654-9792-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCC55528.2022.9912956

***Abstract*—Congestion control algorithms using recurrent neural network (RNN) for bandwidth prediction are expected to improve throughput. Previous studies involving performance evaluations were conducted only using simulated data. However, simulation and real-world environments are largely different and rarely provide equivalent prediction accuracy. Therefore, we will verify whether our proposed method provides better prediction accuracy in a real-world environment.**

We measured communications in a real environment and generated training data by converting packet captured data with measurement of prediction accuracy on the generated data. The results showed that the maximum percentage of correct responses was 79.71%, which was comparable to the results obtained using simulated data.

***Index Terms*—Recurrent neural network, TCP congestion control, Machine learning**

I. Introduction

In recent years, the demand of internet is ever increasing resulting in increase of bandwidth consumption owing to video streaming, teleworking, online gaming, and IoT devices. However, when large amounts of data are transmitted, networks become overloaded and congestion occurs resulting in network delays and other deterioration in communication quality. Congestion control technology avoids congestion and recovers quickly from congestion to prevent such problems.

Various TCP congestion control algorithms studied for congestion control employ the strategy in which the sender increases or decreases the amount of data sent according to the level of congestion to send as much data as possible without causing congestion. One of the typical TCP congestion control algorithms is CUBIC [1], which is a standard algorithm in the Linux kernel and is classifiable as a loss-based algorithm.

Loss-based algorithms use packet loss as an indicator to reduce the amount of transmission. However, the amount of reduction is based on a pre-determined percentage. Therefore, if the amount of reduction is too small, congestion recovery gets delayed or if the amount of reduction is too large, then reduced usage efficiency occurs.

To address this problem, there is a previous study using recurrent neural network (RNN) [2]. The prediction model is developed using learning with network state at one packet loss as a feature and increase or decrease in the transmission volume at the next packet loss as a teacher label. When decreasing the amount of transmission, the algorithm provides the current network state to the model as input and receives the predicted result as output. In that study, experiments using the network simulator ns-3 [3], throughput improvement was confirmed compared to CUBIC.

However, the data collected in the real environment cannot be applied directly, because in the previous research, when data was collected using a simulator, values difficult to obtain in a realistic environment were also used as feature values. For example, in the previous study, the TCP congestion window size was used as a feature value. However, this value cannot be obtained directly from the packet capture results. In addition, the network topology, cross traffic, etc. in the real environment are more complex than those set in the simulation. Therefore, in the real environment, the results may differ owing to factors not assumed in the simulation.

Our study aims to implement RNN congestion control methods in real environments and achieve better performance than existing congestion control methods. Further, we verify the applicability of the proposed method to real-world environments. Therefore, we first collected, analyzed, and transformed communication data in a real network. We measured the prediction accuracy with the model used to train on the real communication data and examined the applicability of our method to real-world environments.

The primary contributions of this study are as follows:

- We measured the prediction accuracy of real-world data used for training.
- We proposed a method to convert real-world packet capture data into feature data used in simulations.
- We discussed future research by classifying the dataset and comparing the throughput and prediction accuracy.

II. Related work

Among several attempts to employ machine-learning techniques in congestion control, many employed reinforced learning [4], [5]. Reinforced learning is a kind of unsupervised learning. Reinforced learning does not require supervision is one of the reasons it is employed. In other words, it reduces human efforts or special settings such as providing teacher data with correct tags (e.g. works only in a simulator).

While reinforced-learning-based methods have several advantages, such as adaptation to newly-emerged situations and no requirement for teacher data; however, it still has unsolved challenges to be implemented and deployed in a realistic environment. Jiang et al. [6], discussed that reinforced-learning-

This work was supported by Toyota Motor Corporation.

based methods require high computational complexity and high memory consumption.

Employing supervised learning in congestion control has not been widely explored yet. To the best of our knowledge, the first attempt to employ supervised learning was conducted by El Khayat et al. [7]. Their target was wireless communication with purpose of using supervised learning to distinguish causes of packet loss, such as link error or congestion.

Bai et al. [2], explored another approach to employ supervised learning in congestion control. They used machine learning to predict success and failure on the recovery size of the congestion window after an event of a packet loss. In their method, teacher data was automatically generated from packet-dump data, which is not easy in general. This feature is important to leverage the value of congestion control with supervised learning in a realistic environment.

III. Overview of each method

For every method described in the following sections, an overview and relationship is provided.

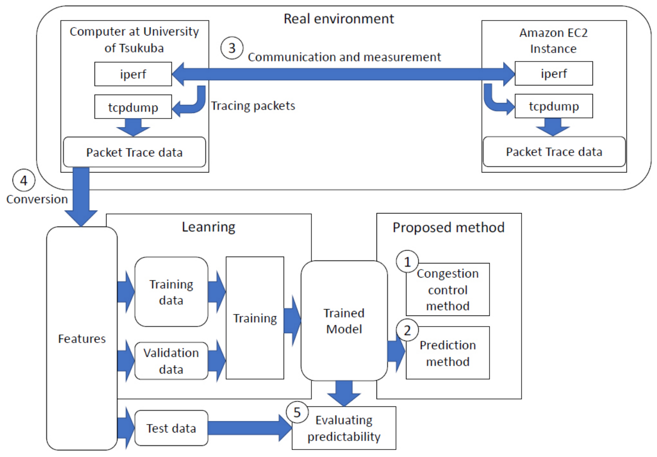

Fig. 1. Prediction overview

Our study consists of three major modules. Figure 1 shows the relationship between each method.

- Congestion control algorithm using prediction in RNN.
- Collection of real-world environmental data.
- Evaluation of Predictive Models.

Figure 1 shows the relationship between each module.

Figure 1, bottom right, the congestion control algorithm proposed in previous studies comprises two methods: (1) a congestion control method to increase or decrease the amount of transmission and (2) a prediction method to determine the amount of decrease in transmission. The model used for the prediction method is generated by learning feature data. The congestion control and prediction method are explained in Section IV, including the details of learning the prediction model.

Upper part of Figure 1, the communication in the real environment is described in Section V, including the communication and collection procedures labeled as (3), the method for converting packet capture data to feature data labeled as (4), and the results of throughput measurement.

Lower part of Figure 1, the evaluation method, labeled as (5), of the prediction model and its results are explained in section VI.

IV. RNN-based congestion control algorithm

A. *Congestion control method*

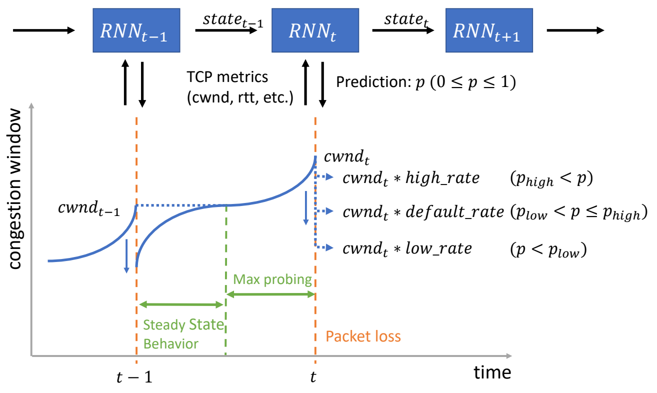
RNN_{t-1}
$state_{t-1}$
RNN_t
$state_t$
RNN_{t+1}
TCP metrics (cwnd, rtt, etc.)
Prediction: p $(0 \le p \le 1)$
congestion window
$cwnd_{t-1}$
$cwnd_t$
$cwnd_t * high\_rate$ $(p_{high} < p)$
$cwnd_t * default\_rate$ $(p_{low} < p \le p_{high})$
$cwnd_t * low\_rate$ $(p < p_{low})$
Max probing
Steady State Behavior
Packet loss
$t-1$
t
time

Fig. 2. The RNN-based predictive approach

The method proposed by Bai et al. is an algorithm based on CUBIC. However, it differs from CUBIC in terms of determining the amount of transmission reduction needed when the amount of transmission needs to be decreased.

CUBIC decreases the amount of transmission when packets are lost. The TCP congestion control algorithm controls the amount of transmission by increasing or decreasing the congestion window size. If there is sufficient bandwidth, the congestion window size is gradually increased to achieve the highest possible throughput. However, if the congestion window size continues to increase, the bandwidth eventually reaches its limit and discards the packets. Loss-based TCP congestion control algorithms, including CUBIC, use discarding of packets as an indicator to decrease the congestion window size. Generally, on arrival of a packet, the receiver notifies the sender by returning an updated acknowledgement (ACK) sequence number. However, if the packet is discarded, the ACK sequence number is not updated or the sender does not receive the ACK The sender side detects packet loss by this method and considers that congestion has occurred. Specifically, CUBIC decreases the congestion window size when three consecutive duplicate ACKs are received or when a retransmission timeout occurs. The congestion is then resolved by temporarily decreasing the transmission volume.

In CUBIC algorithm, a multiplicative decrease factor β determines the amount of congestion window size reduction. When congestion is detected, the congestion window size is reduced by multiplying the congestion window size before reduction by multiplicative decrease factor β. In CUBIC, β is set in advance and remains unchanged during control. In the Linux kernel, the default value of beta is approximately 0.7.

Figure 2 shows the relationship between the prediction and congestion control of the RNN model of the proposed method. When no input/output is performed with the RNN in the upper part of the figure the congestion window size is always reduced to $cwnd * default\_rate$, the proposed method behaves similarly to CUBIC. The proposed method differs from CUBIC in that the value of β is changed in the following three ways.

- If the available bandwidth is expected to be large, set β to a large value ($high\_rate$).
- If the available bandwidth is not expected to change much, β is set to the general value used in CUBIC ($default\_rate$).
- If the available bandwidth is expected to be small, set β to a small value (low_rate).

The prediction result p is output between 0 and 1. p_{high} and p_{low} are thresholds that divide the prediction result into cases. In this case, β is determined by the following formula.

$$\beta = \begin{cases} high\_rate & (p_{high} \leq p) \\ default\_rate & (p_{low} \leq p < p_{high}) \\ low\_rate & (p < p_{low}) \end{cases} \quad (1)$$

The values used by Bai et al. are as follows: $p_{high} = 0.9, p_{low} = 0.7, high\_rate = 0.95, default\_rate = 0.7, low\_rate = 0.5$

B. Prediction method

TABLE I
NETWORK FEATURES.

Name	Explanation
cwnd	Congestion window size at packet loss.
lossint	Time elapsed since the last packet loss event.
rtt	Round-trip time at packet loss.
rttd	Fluctuation value of rtt.
tdup	Whether the packet is lost due to triple duplicate ACKs.
rto	Whether the packet is lost due to retransmission timeout.
lumax	Last cwnd at the time of packet loss that occurred during the CUBIC Max Probing.
luint	Time elapsed since *lumax* was last updated.
ldmax	Last cwnd at the time of packet loss that occurred during the CUBIC Steady State Behavior.
ldint	Time elapsed since *ldmax* was last updated.
tdupint	Time elapsed since the last Triple Duplicate ACK.
rtoint	Time elapsed since the last retransmission timeout.

The proposed method predicts the available bandwidth for the next congestion detection at a given congestion detection. The output/supervisor label is the available bandwidth at the next congestion detection, i.e., whether the congestion window size is increasing or decreasing. In other words, the prediction model performs a binary classification of increasing or decreasing. The teacher label is 1 if the congestion window size is increasing and 0 if it is decreasing. Therefore, the prediction result is closer to 1 if the congestion window size is likely to be increasing and closer to 0 if the congestion window size is likely to be decreasing. Consider the case of learning assuming that the graph at the bottom of Figure 2 is the time variation of the measured congestion window size. The teacher label at $t-1$ is determined by comparing the congestion window size at time $t-1$ and time t. Since $cwnd_{t-1} < cwnd_t$ and the congestion window size is increasing, the teacher label is 1.

The features to be used as input are listed in Table I. cwnd and round trip time (RTT) are basic parameters. A few features are required in relation to Max Probing. Max Probing represents the state in which CUBIC is out of Steady State Behavior and is exploring the upper limit of available bandwidth. Both *lumax* and *ldmax* are congestion window size values. *lumax* is updated in the case of Max Probing and *ldmax* is updated in the case of Steady State Behavior. That is, only one of *lumax* or *ldmax* is always updated. *luint* is the update interval of *lumax* and *ldint* is the update interval of *ldmax*. Whether the current state is Max Probing or not can be determined by comparing the congestion window size. Let $cwnd_t$ be the congestion window size at the time of a prediction and $cwnd_{t-1}$ be the congestion window size one time before that. The conditions and features to be updated are as follows:

- If $cwnd_t < cwnd_{t-1}$, *ldmax* and *ldint* are updated.
- If $cnwd_t > cwnd_{t-1}$, *lumax* and *luint* are updated.

RNN models are effective in predicting the availability of bandwidth because of the time-series nature of the prediction. In a neural network, each feature in the input layer is multiplied by a weight in the intermediate layer and the predicted result is outputted in the output layer. In this experiment, we used a one of the most basic and simple RNN. In addition to feature inputs, simple RNNs also consider the state at the time of one previous prediction as input.

V. MEASUREMENT OF REAL ENVIRONMENT DATA

A. Communication and measurement method

TABLE II
SPECS OF *University Node*.

CPU	Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
Memory	32GB
NIC	Intel Corporation I211 Gigabit Network Connection (rev 03)
OS	Linux4.19.0-16-amd64, SMP Debian 4.19.181-1(2021-03-19)

The measurement environment for the real-world data was an experimental machine installed at the University of Tsukuba (hereinafter referred to as *University Node*) and an Amazon EC2 instance in the Tokyo Region (hereinafter referred to as *EC2 Node*). Communication is performed between *University Node* and *EC2 Node* to collect data. Table II is the respective specifications of *University Node*. The specifications of *University Node* were verified by various commands on the shell. *EC2 Node* is *m6g.xlarge* Instance [8]. *University Node* connects to AWS via the academic information network SINET5 [9]. According to the AWS Japan blog [10], SINET5 and AWS have a peering connection at the IX. This can be confirmed on the PeeringDB between AWS and IX, and between SINET5 and IX on the SINET5 interlink page, respectively. In addition, the above paths follow the actual prepared environment using the traceroute command. Therefore, the lines are shared between SINET5 and AWS. In other words, users mainly within academic institutions affect the communication as cross traffic.

For communication and measurement, iPerf3 [11] and tcpdump [12] are used. Then, the communication by iPerf3 is captured by tcpdump. At this time, tcpdump performs filtering on port number 5201 (iPerf3's communication port number), so that packets other than iPerf3 are not captured. For both *University Node* and *EC2 Node*, CUBIC was used as the TCP congestion control algorithm and the default MTU size was set to 1500bytes. The following procedure is used for each measurement.

1) Start tcpdump and initiate packet capture.
2) Perform communication and measurement for 10 seconds by iPerf3.
3) Stop tcpdump and iPerf3.

The protocol was TCP, with *University Node* as the sender (client) and *EC2 Node* as the receiver (server). Measurements were taken at each hour. Therefore, 24 measurements were taken in one day. Measurements were taken continuously from June 2021 to March 2022. In this study, we focused on four months of data (July, August, September, and October).

We divided the collected dataset into five groups, A - E. A, B, C, and D are the TCP data for each collection month. They are the measured data for July, August, September, and October 2021, respectively. E is the total combined data of A-D. In other words, it is the data for the period of four months.

B. Conversion method

The collected packet capture data is converted to feature data for training. As TCP congestion control is performed on the sender side, packet data captured on the sender side was used. However, as it is difficult to handle raw packet captured data on the conversion program, each parameter of the packet is obtained using the display filter [13] of Tshark [14].

The feature set follows the Bai et al. feature set; however, the packet capture data does not include a congestion window size value. Therefore, we used amount of inflight data (*inflight*) as a substitute for congestion window size. The reason for using *inflight* is that *inflight* represents the amount of data being transmitted and takes a value similar to the congestion window size.

The timing to obtain each feature data is at the instance of each decrease in congestion window size. However, unlike congestion window size, *inflight* value decreases in both cases when packets are lost or when individual packets arrive. Hence, the decrease in *inflight* cannot be regarded as a decrease in congestion window size itself. Therefore, a flag *tcp.analysis.duplicate_ack* is used to estimate when the congestion window size has decreased. When the flag is set, it is assumed that the congestion window size has decreased and one feature data is output. At this point in time, even if the flag stands consecutively, it is assumed to be due to a single packet loss.

C. Results of throughput measurement

TABLE III
STATISTICS OF REAL ENVIRONMENT DATA. DATASET IS NUMBER OF MEASURED FILES.

Group	Duration	Dataset	Throughput[Mbps]			
			min	mean	max	stddev
A	Jul.	742	60.71	500.4	747.3	165.7
B	Aug.	743	86.63	557.7	754.2	107.1
C	Sep.	717	0.1799	497.8	747.0	121.7
D	Oct.	636	2.084	371.8	712.2	154.3
E	Jul. - Oct.	2838	0.1799	485.9	754.2	153.6

As an overall trend of the measured data, the results of the throughput measurement are shown. Table III is the throughput statistics for each group. The DATASET is ideally not the number of days measured x 24 because data with measurement errors are excluded and only the number of valid files are counted.

As E represents the group of A to D combined, the minimum value matches that of C, and the maximum value that of B. The mean value for E is 485.9 Mbps, which is the second lowest value among the five groups, after D. The standard deviation is the third highest among the five groups.

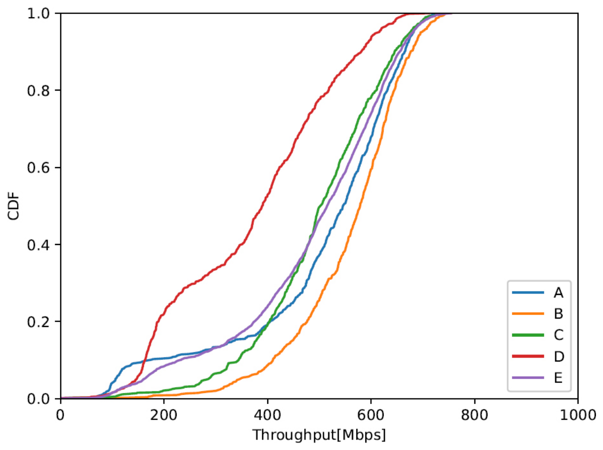

Fig. 3. CDF of each group's throughput

The cumulative distribution function (CDF) in Figure 3 shows a more detailed distribution than Table III. Of particular note are the distributions for A and D. Both have higher standard deviations than B and C. However, while A has about 10% of the data around 100 Mbps, the other data has a similar trend of B and C, with more than 80% of the data distributed around 400-700 Mbps. However, D has an overall left-leaning distribution of throughput, which is generally lower than A, B, and C. In D, more than 20% of the data is distributed at 200 Mbps or less and about half of the data is distributed at 400 Mbps or less.

Figure 4 shows the average throughput for each group at each hour of the day. All groups share relatively high values at 07:00, 08:00, 20:00, and 21:00 hours, and low values at

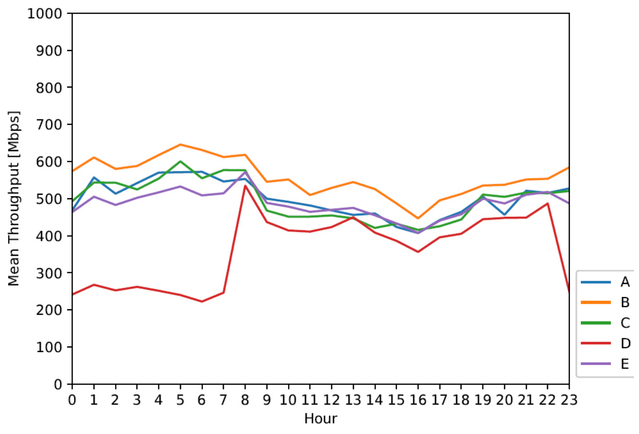

Fig. 4. Average hourly throughput for each group. The horizontal axis is the measurement time and the vertical axis is the average throughput of the data at that time. The time is Japan Standard Time and is written in the 24-hour notation.

16:00 hours. However, Group D's behavior from 23:00 to 07:00 hours clearly differs from that of Groups A, B, and C. While the other groups are stable at around 500 Mbps or more, only Group D has very low values of 200-300 Mbps.

VI. Evaluation of Prediction Accuracy

A. Evaluation method

In the evaluation, the k - fold cross-validation technique was used to reduce bias due to data extraction methods. In this study, k was set to 4. One-fourth of the dataset is used as test data and the other three-fourths for training. Then, another 1/4 of the data for training is used as evaluation data and 3/4 as training data. The features of the test data are input to the prediction model, and the prediction results are calculated by comparing the output of the prediction results with the teacher labels. This procedure is performed a total of $4 \times 4 = 16$ times by changing the range of data used as test and evaluation data, and the average value of 16 times is used as the final result. This reduces the impact of the results on the method of data extraction. Accuracy and receiver operating characteristic area under the curve (ROC-AUC) were used as indicators of prediction accuracy. Both indices indicate that the higher the value, the better the result. Since the prediction results are output between 0 and 1, 0.5 is treated as the threshold value of 0 or 1, and each percentage is calculated by comparing it with the teacher label. To confirm the bias of each data for training, experiments conducted included not only all the data but also the data for each month for training. We performed the implementation in Python using the Python machine learning library, Keras [15].

B. Measurement of simulation data

To measure and compare the prediction accuracy on simulated data, we used the network simulator ns-3 (version 3.24) to collect simulation feature data. The simulation environment was set up based on previous studies. The network topology is a dumbbell topology, where the main traffic and sub-traffic share a bottleneck path with RTT=150ms, Throughput=1Mbps. The main traffic is TCP with CUBIC is the congestion control algorithm. The sub-traffic is UDP communication. The queue used by each node is a drop-tail with a size of 10 packets. The main traffic tries to send as much data as possible. The sub-traffic communicates in the following scenario:

- 20 UDP streams at 10 kb/s that start at random times and last for a uniformly distributed time with Min=10, Max=50
- On-time communication following an exponential distribution with Mean=3, Bound=5, Off-time following an exponential distribution with Mean=30, Bound=50, 20 UDP communications at 50kb/s.

We ran this simulation scenario a total of 100 times with different random seed values.

The 100 files of simulation feature data obtained are designated as Group F. When calculating the prediction accuracy, we use cross-validation as in the case of real measured data.

C. Results of prediction accuracy

TABLE IV
Prediction accuracy. Each column is the group used as training data, accuracy rate, and receiver operating characteristic area under the curve (ROC-AUC).

Group	accuracy	ROC-AUC
A	0.7274	0.7825
B	0.7971	0.8549
C	0.7103	0.7676
D	0.6799	0.7305
E	0.7442	0.8023
F	0.8235	0.8704

Table IV summarizes the average prediction accuracy of the test data for the trained model using each group as training data. Both accuracy and ROC-AUC exhibit a similar trend. The order of prediction accuracy is F, B, (E,) A, C, and D. Group F has the highest prediction accuracy, followed by Group B, and Group D has the lowest prediction accuracy. Group E, using all real environment data, has average results.

VII. Discussion

The prediction accuracy for Group E with real data was 0.7442, which is 0.0793 lower than the prediction accuracy of 0.8235 for simulated data. On the contrary, as a comparison between groups using real data, the difference between groups B and D is 0.1172. As the difference between the real and simulated data is smaller than the difference between real data groups collected in different months, the prediction performance based on real data is practically comparable to the performance based on simulated data.

We next discuss the relationship between throughput and forecast accuracy. Group B has the largest mean throughput, the smallest standard deviation, and the largest prediction accuracy among the real data groups. The CDF and time-specific mean values also confirm that Group B has a stable distribution of data at a relatively high throughput. High throughput means that there is little cross traffic and almost no errors that could cause unexpected throughput drops. Under such conditions, CUBIC is expected to maintain a steady state and the with available bandwidth being almost constant. Therefore, learning and prediction are considered to be relatively easy.

Conversely, D, with lower prediction accuracy, has an average lower throughput and the second largest standard deviation after A. In addition, the number of valid data was relatively low, and many results were errors due to lack of communication. Overall low throughput is widely distributed in CDF as well. The extremely low throughput during the late-night period also did not occur in the other groups. Under such unstable communication conditions, prediction is considered more difficult than under steady-state conditions.

Further investigation of the differences between data with high throughput and prediction accuracy, such as Group B, and data with low throughput and prediction accuracy, such as Group D, could improve prediction accuracy. For example, by measuring the correlation between the distribution of each metric and prediction accuracy as well as throughput, features that are highly useful can be selected. In addition, if the cause of the lower throughput can be identified, other useful features that were not used in this study can be discovered. These investigations and improvements are the subject of future research.

VIII. Conclusion

In this paper, we verified whether the RNN-based prediction method could provide sufficient prediction accuracy in real-world environments. We used packet capture data collected in a real network environment to make predictions with RNN models. Our results showed that the average prediction accuracy was 74.42% when data from the entire period was used for training, which was 7.93% lower than the average prediction accuracy when simulated data was used. This difference is smaller than the difference between the real data and the improvement is realistic. Therefore, same level of performance can be expected in the real environment as that of in the simulation environment.

We also analyzed communication data between machines on campus and Amazon EC2. We confirmed that throughput and prediction accuracy varied during time of day and month of measurement. When only August data was used for training, the average throughput was 557.7 Mbps and the prediction accuracy was 79.71%, both of which were the highest. Conversely, when only October data was used for training, the average throughput was 497.8 Mbps and the prediction accuracy was 67.99%, the lowest of the two.

This implies that the proposed method works effectively in stable networks with high average throughput; however, it shows performance degradation in an unstable network and may not work as effectively as expected. On the other hand, the role of congestion control is to prevent the network from falling into a state of degraded communication quality and to recover promptly when it does fall into such a state. Therefore, further improvement of the prediction accuracy on the dataset with low prediction accuracy indicates future research.

On further analysis of the results of this measurement, optimization of various learning parameters and features, optimization of training data selection methods, and preprocessing, we expect to improve prediction accuracy. In addition, a more practical evaluation is possible by implementing the proposed method on a Linux kernel and measuring and comparing the throughput. We plan to improve the prediction accuracy and implement and measure the proposed method on the Linux kernel as future research.

Acknowledgement

We would like to express our gratitude to Takuya Hamasaki and Yosuke Inoue for their contributions in the early stage of this research when they were students at the University of Tsukuba. Further, we thank Toyota Motor Corporation for supporting this work.

References

[1] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[2] 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.

[3] "ns-3 — a discrete-event network simulator for internet systems," https://www.nsnam.org/, (Accessed on 04/01/2022).

[4] 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.

[5] 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. [Online]. Available: https://proceedings.mlr.press/v97/jay19a.html

[6] H. Jiang, Q. Li, Y. Jiang, G. Shen, R. Sinnott, C. Tian, and M. Xu, "When machine learning meets congestion control: A survey and comparison," *Computer Networks*, vol. 192, p. 108033, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621001407

[7] I. El Khayat, P. Geurts, and G. Leduc, "Enhancement of tcp over wired/wireless networks with packet loss classifiers inferred by supervised learning," 2010.

[8] "Amazon EC2 Instance Types - Amazon Web Services," https://aws.amazon.com/ec2/instance-types/, (Accessed on 03/30/2022).

[9] "Science Information NETwork 5," https://www.sinet.ad.jp/en/top-en, (Accessed on 03/30/2022).

[10] "Using AWS via SINET5 at academic research institutes. [Translated from Japanese.]," https://aws.amazon.com/jp/blogs/news/sinet5-aws-explain/, 19 Jun 2019, (Accessed on 03/30/2022).

[11] "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool," https://iperf.fr/, (Accessed on 03/30/2022).

[12] "Home — TCPDUMP & LIBPCAP," https://www.tcpdump.org/, (Accessed on 03/30/2022).

[13] "Wireshark - Display Filter Reference," https://www.wireshark.org/docs/dfref/, (Accessed on 03/30/2022).

[14] "tshark," https://www.wireshark.org/docs/man-pages/tshark.html, (Accessed on 03/30/2022).

[15] "Keras: the Python deep learning API," https://keras.io/, (Accessed on 03/30/2022).