2ND WORKSHOP
"MACHINE LEARNING & NETWORKING" (MaLeNe)
PROCEEDINGS

SEPTEMBER 4, 2023

CO-LOCATED WITH
THE 5TH INTERNATIONAL CONFERENCE ON NETWORKED SYSTEMS (NETSYS 2023)
POTSDAM, GERMANY

In-Network Round-Trip Time Estimation for TCP Flows

Daniel Stolpmann, Andreas Timm-Giel
Hamburg University of Technology, Institute of Communication Networks, Germany
{daniel.stolpmann, timm-giel}@tuhh.de

***Abstract*—Transmission Control Protocol (TCP)'s tendency to fill up buffers in the network results in long standing queues. Active Queue Management (AQM) tries to solve this issue but typically requires manual tuning as the optimal parameters depend on the Round-Trip Time (RTT) of the flow, which is unknown to the AQM. In this paper, we use network simulation and supervised learning to train a neural network to infer the RTT of a TCP flow from its queuing behavior. We transfer our model into a real-time network emulator and show that it is able to estimate the RTT with an error of only a few milliseconds.**

***Index Terms*—Active Queue Management, Machine Learning, Dataset Generation, Supervised Learning, Network Emulation**

I. Introduction

Transmission Control Protocol (TCP)'s tendency to fill up the buffer in front of the bottleneck link often results in a long standing queue also known as bufferbloat. Active Queue Management (AQM) tries to solve this issue by prematurely dropping or, if combined with Explicit Congestion Notification (ECN), marking packets to signal congestion to the Congestion Control (CC) algorithm at the sender. However, a common problem of AQM algorithms is the necessity to tune their parameters to the operating conditions. In [1], it was shown that the optimal buffer size for a TCP flow that results in full link utilization and minimal queuing delay depends on its base Round-Trip Time (RTT) (without queuing), which is unknown at the bottleneck link.

In this paper, we present a Machine Learning (ML) model that infers the base RTT of a TCP flow from its queuing behavior. Our approach is based on an efficient way to generate training data using a high-level network simulation, which is used to train a neural network using supervised learning. Finally, we transfer the model into a real-time network emulator to evaluate its performance on real traffic.

II. Related Work

In [2], the RTT of a TCP flow was estimated based on the timestamps in the TCP header using ML. In [3], the size of the buffer at the bottleneck link was adapted for a TCP flow depending on its RTT and the used CC scheme to achieve high throughput and low delay based on queue statistics using Deep Reinforcement Learning (DRL).

III. System Model

Our training data is generated by a network simulation implemented using Python and SimPy[1]. The simulation models a general communication network with a packet sender and a receiver that are connected via a full-duplex connection. The sender performs CC with a slow-start and a congestion avoidance phase as in TCP. In each direction, a configurable delay of half the base RTT is added to the packets. In the direction from the sender to the receiver, there is a packet queue. Packets are dequeued at a configurable time interval to simulate a link with a fixed capacity. Whenever the queue exceeds the buffer size or the queuing delay exceeds a controllable threshold, packets are marked as in ECN. The marking of a packet is signaled back from the receiver to the sender, which sees it as a congestion signal and reduces the Congestion Window (CWND) accordingly. The model is depicted in Figure 1.

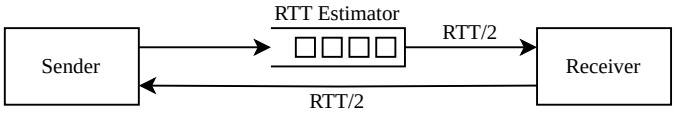

Fig. 1. Illustration of the system model

At the queue, the link capacity, the link utilization, the queuing delay and the queue length are measured every $10\,\mathrm{ms}$. To obtain the link capacity and the throughput, the number of transmission opportunities and transmitted packets are counted over the same interval. To get the queuing delay, it is calculated as the queue length divided by the link capacity [4] instead of timestamping each packet. As input for the estimator, a history of the values at the last 200 time steps is used. Additionally, the differentials $\Delta x_t = x_t - x_{t-1}$ are calculated for every value and also included in the input. To make the trained model robust against real-world imperfections, jitter is added to the measuring interval as well as to the packet pacing at the sender.

The estimator neural network is created and trained using Keras[2]. It has an input layer with 1600 neurons followed by two fully connected hidden layers with 256 neurons each using the Rectified Linear Unit (ReLU) activation function. Batch normalization is applied at the input, while layer normalization is performed after each hidden layer. Finally, the network condenses to a single neuron with linear activation in the output layer to estimate the base RTT.

[1]https://simpy.readthedocs.io (Accessed: 03.07.2023)

[2]https://keras.io (Accessed: 03.07.2023)

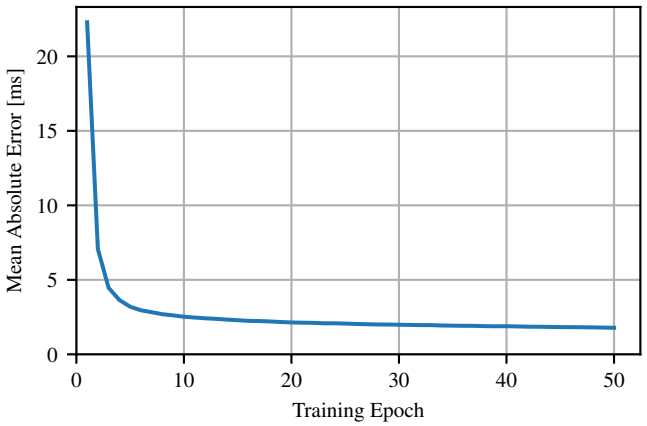

Fig. 2. Error in the base RTT estimation over the training epochs

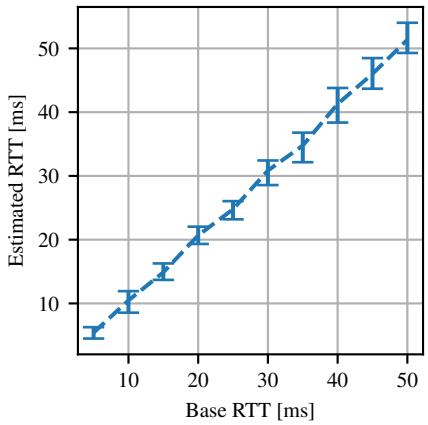

Fig. 3. Estimated RTT over the configured base RTT in the emulator

IV. Training

A. Dataset Generation

Multiple simulations are executed in parallel using the Ray framework[3] to create the dataset. Each training batch contains the data from one step in each simulation, resulting in a batch size of 128 with an equal number of simulations. The simulations are executed in episodes of 30 s. In the beginning, each simulation is executed for a random time of up to the length of the first episode where no samples are collected to desynchronize the simulations. At the start of each episode, a random base RTT between 1 and 60 ms and a queuing delay threshold between 0 and 100 ms is chosen. During the episode, the link capacity is resampled every 1 to 10 s to be between 100 and 2500 packets/s. Varying these three parameters creates a diverse training dataset for different base RTTs, link capacities and buffer sizes. The dataset contains 10 000 batches and thus a total of 1 280 000 training samples.

B. Supervised Learning

The neural network is trained supervised using randomly sampled training batches from the dataset. The queue statistics are used as the input and the base RTT as the label. The Mean Squared Error (MSE) is used as the loss function. The length of each training epoch corresponds to the size of the dataset and the network is trained for a total of 50 epochs using the Adam optimizer with a learning rate of 10^{-3}.

The estimation error over the training epochs is shown in Figure 2. It can be seen that the Mean Absolute Error (MAE) starts at around 23 ms, but goes down quickly and reaches a value of around 2 ms at the end of the training.

V. Evaluation

To evaluate the performance of the RTT estimator with a real TCP flow, it is implemented as a module in the FlowEmu network emulator [5]. The module acts as a packet queue and uses the TensorFlow C++ Application Programming Interface (API)[4] to load the trained neural network. The output of the neural network is filtered using an Exponentially Weighted Moving Average (EWMA) with a weight of 0.1 for the new sample. The emulated network corresponds to the one described in Section III, but iPerf[5] with TCP New Reno is used as the sender and the receiver. The buffer size is limited to 80 packets and the link capacity is set to 2000 packets/s.

The estimated RTT over the configured base RTT is shown in Figure 3. The plot shows the mean value over an experiment duration of 60 s, where the first 5 s are omitted as warm-up period. The bars mark the 5% and 95% percentiles, respectively. It can be seen that the estimated RTT is close to the configured base RTT with an error of only a few milliseconds.

VI. Conclusion

We trained a neural network on simulation data to estimate the base RTT of a TCP flow at the bottleneck link and showed its accuracy in a real-time network emulator. The estimated RTT can be used in future work to tune the parameters of classic AQM algorithms or as input for novel DRL approaches.

References

[1] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, Oct. 2004.

[2] D. H. Hagos, P. E. Engelstad, A. Yazidi, and C. Griwodz, "A Deep Learning Approach to Dynamic Passive RTT Prediction Model for TCP," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, London, England, UK, Oct. 2019.

[3] M. Bachl, J. Fabini, and T. Zseby, "LFQ: Online Learning of Per-flow Queuing Policies using Deep Reinforcement Learning," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, Sydney, NSW, Australia, Nov. 2020, pp. 417–420.

[4] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "ABC: A Simple Explicit Congestion Controller for Wireless Networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*, Santa Clara, CA, USA, Feb. 2020, pp. 353–372.

[5] D. Stolpmann and A. Timm-Giel, "FlowEmu: An Open-Source Flow-Based Network Emulator," *Electronic Communications of the EASST*, vol. 80: Conference on Networked Systems 2021 (NetSys 2021), Sep. 2021.

[3] https://www.ray.io (Accessed: 03.07.2023)

[4] https://www.tensorflow.org/api_docs/cc (Accessed: 03.07.2023)

[5] https://iperf.fr (Accessed: 03.07.2023)