# DeePCCI: Deep Learning-based Passive Congestion Control Identification

Constantin Sander
RWTH Aachen University
sander@comsys.rwth-aachen.de

Jan Rüth
RWTH Aachen University
rueth@comsys.rwth-aachen.de

Oliver Hohlfeld
Brandenburg University of Technology
oliver.hohlfeld@b-tu.de

Klaus Wehrle
RWTH Aachen University
wehrle@comsys.rwth-aachen.de

## ABSTRACT

Transport protocols use congestion control to avoid overloading a network. Nowadays, different congestion control variants exist that influence performance. Studying their use is thus relevant, but it is hard to identify which variant is used. While passive identification approaches exist, these require detailed domain knowledge and often also rely on outdated assumptions about how congestion control operates and what data is accessible. We present DeePCCI, a passive, deep learning-based congestion control identification approach which does not need any domain knowledge other than training traffic of a congestion control variant. By only using packet arrival data, it is also directly applicable to encrypted (transport header) traffic. DeePCCI is therefore more easily extendable and can also be used with QUIC.

## CCS CONCEPTS

• **Networks** → **Transport protocols**; **Network resources allocation**; **Network measurement**; Packet classification; • **Computing methodologies** → *Supervised learning*; *Neural networks.*

## KEYWORDS

Deep Learning, Congestion Control, Congestion Control Identification, Passive Measurements

**ACM Reference Format:**
Constantin Sander, Jan Rüth, Oliver Hohlfeld, and Klaus Wehrle. 2019. DeePCCI: Deep Learning-based Passive Congestion Control Identification. In *NetAI '19: ACM SIGCOMM 2019 Workshop on Network Meets AI & ML, August 23, 2019, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341216.3342211

## 1 INTRODUCTION

Congestion control (CC) [12] is a fundamental building block in today's transport protocols and strongly influences the performance of data transmissions. Initially built in the 1980s to counteract the congestion collapse of the early Internet [17], CC still evolves and new variants, such as BBR [1] or Vivace [3], emerge.

CC introduces a congestion window (cwnd) that limits the number of unacknowledged bytes in flight. Each CC algorithm defines how it governs the evolution of the cwnd subject to algorithm-specific congestion signals. Given the number of CC approaches and their (inter-) performance implications [9], it is hence relevant to study the CC usage. E.g., it is easier to tune a new CC for fairness, if it is known which other algorithms it typically competes with.

However, existing works (e.g., [2, 18, 24]) for identifying CC variants are not adapted to the latest CC and transport protocol developments. Extending and maintaining these approaches is complex since it requires detailed domain knowledge to know how CC parameterization and configuration affect their behavior. This becomes even more critical when CC leaves the kernel and is introduced in user space protocols such as QUIC [11] that are comparably easy to change and see already large-scale deployment [21]. Moreover, many identification approaches are based on fragile assumptions. For example, they fail when using TCP pacing, e.g., in combination with RENO [12] or CUBIC [5]. Aggravatingly, all passive approaches known to us are based on the assumption of parsable header information. A fully encrypted transport, as for example QUIC implements, renders these designs invalid and would require significant changes if possible at all. Thus, it is currently challenging to reason about the CC deployment.

As a first step to tackle these challenges, this paper presents DeePCCI, a supervised deep learning-based approach for passive congestion control identification. It identifies CC variants solely based on flow packet arrival time information and thus even works on encrypted transport headers. Moreover, it uses deep learning to learn features — thereby avoiding manual, domain-specific feature-engineering. Thus, unlike related approaches, DeePCCI makes no assumptions other than the availability of flow packet timings nor requires any domain knowledge other than being able to gather training traffic of a CC variant. We argue that this assumption and hand-tuning free method allows for generic and extensible CC identification in Internet traffic.

Specifically, we present DeePCCI's design, its evaluation, and its limitations and make the following contributions:

- We describe the preprocessing of traffic and the deep learning model for identification of congestion control variants.
- We present how to train the model for multiple congestion control variants with labeled data generated in a testbed.
- We evaluate the performance in a testbed for CUBIC, RENO, and BBR as major congestion control variants. We show that the approach is able to identify flow congestion control variants

*NetAI '19, August 23, 2019, Beijing, China*

in various scenarios, but we also present and discuss scenarios where it is unable to identify congestion control variants.

**Structure.** Section 2 discusses the state-of-the-art in CC identification and its shortcomings. Section 3 describes the design of DeePCCI while Section 4 shows how we generate training data and evaluate our approach. Finally, Section 5 concludes the paper and discusses future work. Our paper does not raise any ethical issues.

## 2 RELATED WORK

Various works deal with identifying CC variants. There are two main categories for these approaches: Identification approaches using passive [2, 6, 13, 18, 20] or active [19, 24] measurements.

**Active approaches** stimulate CC reactions for detection by actively opening connections and manipulating them. Padhye and Floyd propose TBIT [19] which sends crafted TCP segments to web servers to actively trigger congestion control. It records which segments are sent in reaction to a lost packet, as this reaction differs drastically between the CC variants distinguished in TBIT.

Yang et al. present CAAI [24] extending TBIT's approach. CAAI artificially delays ACKs observing all in-flight segments in order to estimate the cwnd of the sender. CAAI then induces packet loss and extracts characteristic features from the changing cwnd. These features are subsequently used for classification using random forests. While both approaches achieve high identification accuracies, relying on active measurements can easily introduce a measurement bias due to wrongly selected hosts.

**Passive approaches** (as ours) do not interact with hosts and rather rely on traffic traces to infer the used CC variants of traffic flows. Therefore, they allow gathering information on real traffic which depends on vantage points, not actively selected hosts.

Paxson et al. and Jaiswal et al. rebuild TCP state machines with tcpanaly [20] and tcpflows [13] to compare received with expected packets. Both approaches require very detailed CC and even implementation knowledge to rebuild the state machines. Our approach differs in that it does not need detailed CC knowledge.

Casagrande et al. [2] use characteristic changes in a cwnd as features in their approach TCPMoon. Different handcrafted rules are checked against these features to distinguish CCs. For cwnd estimation, the authors use an RTT estimation based on TCP timestamps. Identifying flows without the TCP timestamp option or when transport headers are encrypted is hence not feasible with TCPMoon. As our approach does only observe the behavior of packet arrivals, it does not need any cleartext transport protocol fields.

Oshio et al. [18] propose a clustering-based method. They extract features of a cwnd based on an RTT estimate and cluster these to discriminate two competing CC variants. Our approach differs in that it is not limited to only two competing variants.

Hagos et al. [6] use the outstanding bytes between sender and receiver as a rough and noisy cwnd estimate. This estimate is refined using a recurrent neural network. Sudden decreases in the refined cwnd are used as an estimate of the multiplicative decrease factor which differs between CUBIC, BIC and RENO. While this approach is similar in that it uses deep learning, it still requires the manually engineered multiplicative decrease factor and thus only identifies loss-based CC. Our approach uses an end-to-end deep learning model, identifies also delay-based CC and avoids manual features.

## 3 DeePCCI DESIGN

In this section, we present the architecture of DeePCCI. We show our design goals and list the steps necessary to identify CC variants without variant-specific domain knowledge.

### 3.1 Design Goals

DeePCCI is designed to allow passive CC variant identification. It uses supervised deep learning to learn features of CC variant behavior from generated traffic flows and subsequently allows to identify / classify the used CC variants of new flows with these features. Our design is guided by the following two main points:

- First, we want to avoid assumptions such as the availability of headers or specific CC behavior. Such assumptions are prone to change or break with slight changes to the algorithms and may render the approach inapplicable. We, therefore, do not analyze packet headers, as the assumption of unencrypted packet headers will not hold in the advent of QUIC. Moreover, we do not need the ACK control-flow from receiver to sender, as it might be unavailable due to routing asymmetry and encryption.
- Second, we want to manually engineer as few features as possible. Otherwise, new CC variants would require new manually crafted domain-specific features, so having the ability to automatically learn which features to extract for classification will enable future applicability and easier adaption in case of changes.

We argue that these guidelines enable our approach to be robust against minor changes in the Internet and to be easily extendable for major changes by retraining on new data.

### 3.2 CC Identification & Learning Approach

We now present the architecture of DeePCCI along Figure 1.

**CC Manifestation in Traffic.** We use passive traffic captures to identify the CC variant. Our only input is the packet arrival time of a flow, we thus only assume packet timing information (e.g., unlike Netflow) and that we are able to associate packets to flows. Since any CC variant will effectively control the sending of packets in terms of how many will be sent and at which point in time, we argue that packet timing data inherently captures a CC's behavior. We associate the packet arrivals into constant-sized bins to get a histogram of packet arrivals with equidistant timesteps for a fixed timing structure. We denote this histogram as $X = [x_0, ..., x_t]$ where every $x_i$ is a one-dimensional feature. The timing structure is then exploited by the convolutions of our neural network.

**Packet Arrival Times as the Only Feature.** We feed the histogram of packet arrivals $X$ into a deep neural network (DNN). Our primary motivation for using deep learning is its ability to learn features from data [16]. Therefore, we do not engineer any features other than packet timing—the signature of every CC—to ensure the versatility of our approach.

**Deep Learning-based Classification.** Inspired by convolutional, long short-term memory DNN (CLDNN) approaches used in speech recognition [22], our DNN consists of a convolutional neural network (CNN) [16] and a long short-term memory (LSTM) [8] part (see lower part of Figure 1). First, the histogram is processed by a CNN which we regard as a feature extraction stage and is derived from a 2D VGGNet-13 [23] which is mainly used for image recognition. We use 5 subsequent modules, which consist of 1D

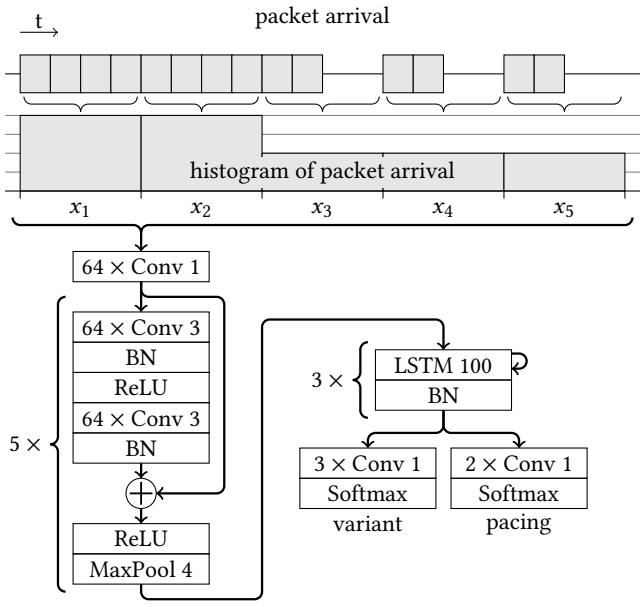



**Figure 1: Architecture of DeePCCI. Captured packets are at first assigned to histogram bins with regard to arrival time. The packet arrival histogram is then used in a 1D-CNN with added unidirectional LSTM RNN layers and a final splitted classification layer.**

convolutional layers, ReLU activations and 1D maxpooling layers. We combine two 1D convolutional layers with 64 filters of size 3 and stride 1 with a ReLU activation in between. We then add the input to the second convolutional layer output as identity skip connection and add a final ReLU activation. To be able to add the input to the output w.r.t. the dimensions, we add a first convolution with 64 filters of filtersize 1 to increase the dimensionality of the histogram input. We adopted the skip connections from the residual learning [7] approach, as it allowed our network to train more easily and achieve better inference results. In contrast to residual networks, we use a maxpooling layer instead of convolutions with higher strides to reduce the dimensionality after each module, as our convolutions of size 3 are smaller than the pooling strides of 4 we are using. Moreover, we use a batch normalization [10] layer after every convolution to normalize the activations in turn reducing overfitting and improving training speed. The CNN part of the network reduces the incoming data dimensionality by a factor of 1024. 1024 timesteps (e.g., 1024ms with 1ms binsize) are reduced into a single LSTM timestep. This timestep is then used in three subsequent unidirectional LSTM [8] layers with 100 memory units. If only fixed sequences of packets should be identified, it proved to be sufficient to also use a fully connected layer as a standard VGGNet does. However, as we are interested in identifying varying length traffic flows, we use LSTM recurrent neural network (RNN) layers to build up a memory depending on previous behavior. Following every LSTM layer, we again incorporate a batch normalization layer for the same reasons as before. After the last LSTM layer, we compute 3 + 2 logits using convolutional layers with filtersize 1. We split the logits for two different softmax classifications: 3 logits are used for CC classification (e.g., RENO, CUBIC, BBR), while 2 logits are used for classification whether pacing was used. This allows us to split the loss function into two crossentropy-losses which we can weight accordingly to emphasize correct CC variant rather than correct pacing classification during training.

| | |
|---|---|
| **Bandwidth** (Mbps) | 1, 2,..,10, 20, 25, 30, 40, ..., 100, 150, 200 |
| **Latency** (ms) | 1, 2, ..., 20, 30, ..., 100, 150, ..., 300 |
| **BDP Factor** | 0.5, 1, 2, 5, 10 |

**Table 1: Parameters for the single-host network**

| | |
|---|---|
| **Bandwidth** (Mbps) | 1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 40, 50, 100 |
| **Latency** (ms) | 1, 2, 5, 10, 20, 50 |
| **BDP Factor** | 1, 5, 10 |

**Table 2: Parameters for the multi-host network.**

## 4 EVALUATION

We next evaluate DeePCCI in a testbed setup by focusing on CUBIC, RENO, and BBR as major CC variants. We first describe our setup used for training data generation and testing. Subsequently, we present how well DeePCCI identifies CC variants and put a special focus on where it struggles.

### 4.1 Experimental Setup

Since DeePCCI bases on supervised deep learning, we need labeled data of traffic flows in order to train the neural network. Labeled data of traffic flows with respect to CC is scarce, so we generate this data on an experimental testbed.

**Mininet-based Network Testbed.** We utilize Mininet [15] to generate traffic in two topologies subject to different network conditions. We vary the number of TCP senders, the link latency, the bottleneck link bandwidths, and its queue sizes defined as multiples of the bottleneck link's BDP. The senders send 60s of fully-loaded TCP streams using a chosen CC variant (in this paper, we focus on BBR, CUBIC, RENO - when using pacing we note "-p" to the name) from a standard Linux 4.18 kernel. Other senders than the to be observed sender (if available) start 2s prior. In each setting, we capture traffic before and after the bottleneck link in the respective network that is used to train and evaluate our approach.

**Single-Host Network.** The single-host network serves as a baseline condition where only a single TCP sender is active—the simplest condition for CC detection. It consists of a dumbbell topology where the sending host is connected to a router which is connected via a bottleneck link to a router which is connected to another host. No background traffic is generated. The specific parameters used in this network topology are shown in Table 1.

**Multi-Host Network.** More complex than the single-host network, here, three hosts reside on each side of the network. The hosts use all possible combinations of the CC variants such that we generate traffic with impact of all other CC variants. We reduced the parameter set to counter the growing parameter space for all combinations. The specific parameters used are shown in Table 2.

**Cross-Traffic Network.** In the cross-traffic network, we emulate the influence of 3 side flows which cross 4 main flows on parts of

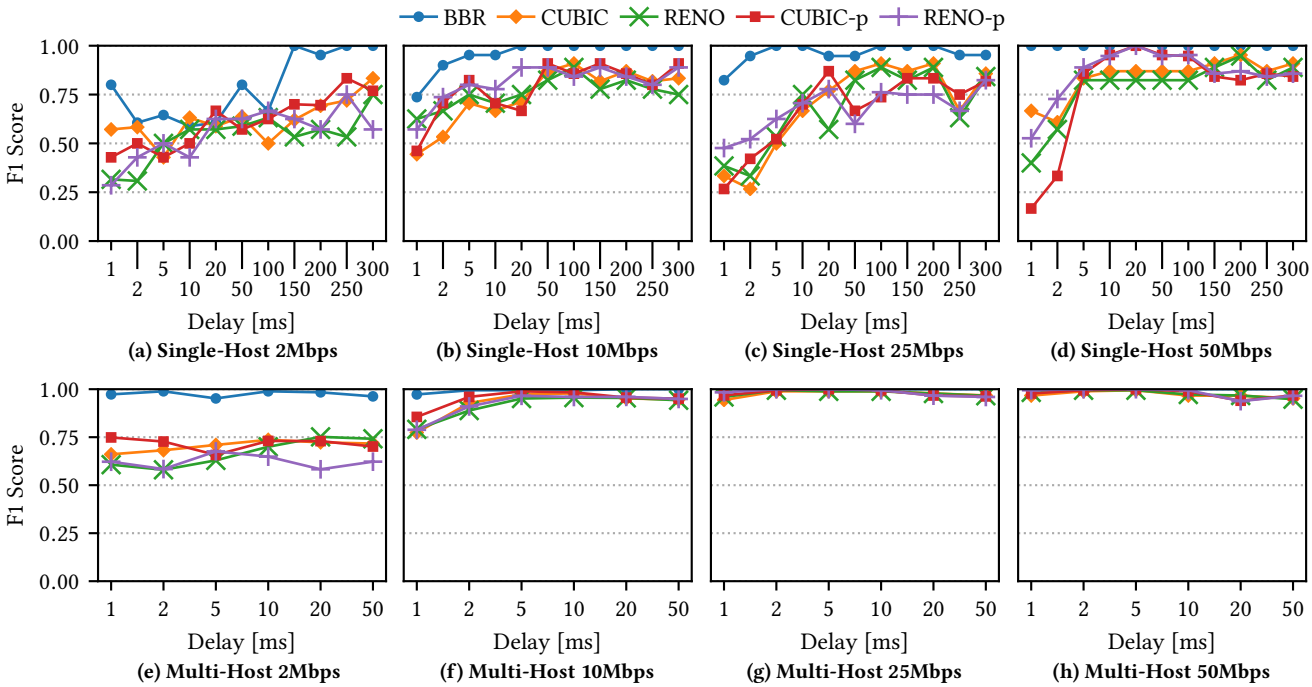



**Figure 2: F1 scores for single and multi-host network w.r.t. bandwidth and delay.**

a route. The side flows traverse a 25Mbps, a shared 50Mbps and another 25Mbps link. The main flows traverse 3 links limited to 50Mbps. The second link is shared with the side flows in the same sending direction. The latency of all links is fixed to 10ms, while the queue sizes can be either 1, 5 or 10 times the BDP. The side flows and 3 main flows use CUBIC, RENO, and BBR as CC to emulate background traffic. The fourth main flow uses a CC of choice. We capture traffic before and after the shared link. Thus, this network shows the influence of vantage points where not all traffic is visible and where CCs with different bottlenecks interact.

**Training.** As training data, we use the single and multi-host network datasets with vantage points before and after the bottleneck. We utilize bottleneck link rates of 4Mbps and 30Mbps as the validation set on which we optimized the DNN and 2Mbps, 10Mbps, 25Mbps and 50Mbps bottlenecks as the test set. This fixed holdout allows us to use the same trained model during all of the evaluation and to reason more in detail about certain results.

We use 1ms as histogram bin-size such that the histogram contains 60000 entries after 60s of traffic. For training, we utilize back-propagation through time unrolling all LSTM timesteps. Moreover, we train the model with the Adam[14] optimizer with a learning rate of 0.001 and a learning rate schedule to halve it every 5 epochs, use a batch size of 32 and early stop after 5 epochs of no validation loss improvement. As loss, we use the 1:4 weighted cross-entropy between pacing and variant classification averaged over all timesteps and batch entries (to put more emphasis on the variant).

## 4.2 DeePCCI Performance

*4.2.1 Identification Performance by Delay and Bandwidth.* At first, we evaluate the identification performance w.r.t. delay and bandwidth. For this, we evaluate the F1 scores of all variants per bandwidth and delay for the single and the multi-host network before and after the bottleneck. The F1 scores are shown in Figure 2.

We can see that our approach distinguishes very well between BBR and loss-based congestion for larger bandwidths above 10Mbps. Bandwidths $\geq$ 10Mbps and delays $\geq$ 5ms lead to individual F1 scores above 90% in the more realistic multi-host scenarios and F1 scores above 55% in the single-host scenario while BBR is identified with F1 scores above 90%. Including also the smaller bandwidths, the minimum F1 score drops onto 40% in the single-host case and 55% in multi-host case. Including also the smaller delays decreases the performance for the single-host network further with F1 scores below 20%. In general, we can see that larger bandwidths, larger delays and multiple competing flows are beneficial for our approach.

We attribute the effect of the bandwidth to the integer discretization of the congestion window. The maximum cwnd depends on the bottleneck bandwidth. Therefore, more steps of, e.g., the cubic behavior of CUBIC are sampled with larger bandwidths. Lower bandwidths imply fewer sampled steps and the cubic behavior is harder to discriminate against, e.g., linear behavior as with RENO.

Next, we could see that higher delays increased the identification performance. We attribute the effect of the delay to our histogram bin-size. If the delay is too small, the change in packet arrival is too fast for the time sub-sampling of the bins such that the behavior cannot be derived if it does not differ drastically enough (e.g. with higher bandwidths). Moreover, the delay also affects the decision

of whether pacing was used due to too many packets (being paced or not) falling into the same bin (not shown).

However, we can see that also the multi-host case achieves better results for the same delays. We attribute this effect to the competition of the flows. Beside filling the queues, increasing the queuing delays and leading to more congestion, these hosts also imply a competition for packets in the queue. While the rate of a single-host flow quickly does not increase with an increasing cwnd when the link is saturated, increasing the flow's cwnd in the multi-host scenario can increase the share of its packets in the queue in turn increasing its rate. The individual changes to the cwnd of the different congestion control variants therefore have a higher influence onto the rate which affects the packet-arrival stronger and over a longer time contrasting the issues of smaller delays and bandwidths.

As we have seen, bandwidth and delay impact our approach and too small delays / bandwidths result in low identification performance. For our further evaluation we will continue with 50Mbps as bandwidth to better see where the approach is further challenged.

*4.2.2 Vantage Point before Bottleneck.* We now evaluate the identification performance of our model with only a vantage point *before* the network bottleneck. Here, we see packets *before* they are dropped due to congestion or shaped due to queues and thus capture packet arrival times as generated by CC. We, therefore, expect very good identification results.

**Single-Host Network.** The single-host identification performance before the network can be seen in Table 3 (values not in brackets). The upper results in a cell were evaluated on the same delays as for the multi-host scenario (mh), while the lower results use all available delays in the test-set (all). We employ this split as our single-host test set contains more delays but we would like to maintain comparability. As expected, the results are good with an F1 macro average and overall accuracy of 89% when observing the same delays as in the multi-host network. With all delays, the accuracy increases to 98% due to the inclusion of higher delays.

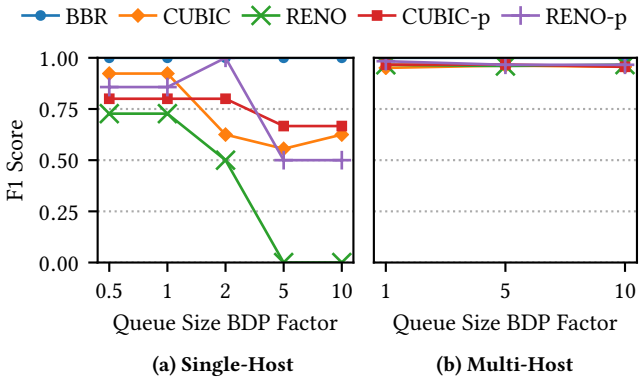
**Multi-Host Network.** The multi-host (i.e., when having competing traffic of other hosts) identification performance is shown in Table 4 (values not in brackets). The results are significantly better as already observed before with an overall accuracy of 99% and a macro F1 average of 99%. Comparing the single and multi-host performance, we can see a significant drop in recall for CUBIC-p and decreased precision for CUBIC in the single-host scenario. Moreover, also precision and recall are smaller for RENO and RENO-p. Indeed, CUBIC-p flows are often erroneously classified as CUBIC and RENO flows are confused with RENO-p and vice versa (not shown due to space limits). This also explains why higher delays are beneficial, as this way the pacing is easier to distinguish.

*4.2.3 Vantage Point after Bottleneck.* When applied to realistic Internet traffic, CC identification must be robust to packet arrival patterns being altered by queues and other elements. We therefore now evaluate the performance on traffic *after* being shaped by the bottleneck link on the same trained model as before.

**Single-Host Network.** For the single-host network, we can see some interesting differences. The identification performance is again shown in Table 3 (values in brackets). Our approach is more challenged with data after the bottleneck which can be easily explained by the queuing dynamics, smoothing the behavior, making

| | | Precision | Recall | F1 | Accuracy |
|---|---|---|---|---|---|
| BBR | mh | 1.00 (1.00) | 1.00 (1.00) | 1.00 (1.00) | 1.00 (1.00) |
| | all | 1.00 (0.99) | 1.00 (1.00) | 1.00 (1.00) | 1.00 (1.00) |
| CUBIC | mh | 0.78 (0.59) | 0.97 (0.90) | 0.87 (0.71) | 0.94 (0.85) |
| | all | 0.95 (0.63) | 0.99 (0.92) | 0.97 (0.75) | 0.99 (0.88) |
| RENO | mh | 0.90 (0.77) | 0.87 (0.33) | 0.88 (0.47) | 0.95 (0.85) |
| | all | 0.98 (0.89) | 0.97 (0.47) | 0.98 (0.61) | 0.99 (0.88) |
| CUBIC-p | mh | 0.96 (0.83) | 0.73 (0.67) | 0.83 (0.74) | 0.94 (0.91) |
| | all | 0.99 (0.91) | 0.93 (0.81) | 0.96 (0.85) | 0.98 (0.94) |
| RENO-p | mh | 0.87 (0.85) | 0.90 (0.73) | 0.89 (0.79) | 0.95 (0.92) |
| | all | 0.96 (0.90) | 0.98 (0.88) | 0.97 (0.89) | 0.99 (0.95) |
| | mh | Overall Accuracy: | | 0.89 (0.73) | |
| | all | | | 0.98 (0.81) | |

**Table 3: Identification metrics for single-host network before (after) 50Mbps bottleneck for all delays or delays as in the multi-host network (mh) for comparability.**




**(a) Single-Host** **(b) Multi-Host**

**Figure 3: F1 scores for single and multi-host network after a 50Mbps bottleneck w.r.t. queue size for the same delays as in the multi-host network for comparability.**

it harder to identify certain characteristics of CC. While most recalls decrease slightly, RENO's recall decreases to 33%. We found a correlation between the queue size (as a factor of the BDP) and RENO's F1 score, which is shown in Figure 3a. For BDP factors 0.5 and 1, RENO is classified with F1 scores above 70% but for greater factors, it reduces to 50% for twice of the BDP and 0% for 5 and 10 times the BDP. In fact, many of the RENO flows are classified as CUBIC (not shown), which also results in worse precision and hence worse F1 score for CUBIC.

Our approach, therefore, seems to be prone to bufferbloat [4] w.r.t. RENO. We account this effect to the shaping nature of queues, which in essence paces out data smoothly when the link is saturated causing RENO as well as CUBIC to look more similar. This also explains why paced RENO can be distinguished better: Packets are more equally distributed over an RTT and there is an increased chance that the queue is already drained if the next packet arrives, so the queue size is less impacting the smoothing.

**Multi-Host Network.** When evaluating the multi-host network, we see more expected results. The identification performance is again shown in Table 4 (values in brackets). The overall accuracy reduces only insignificantly by 2 percent-points onto 97%. We are

| | Precision | Recall | F1 | Accuracy |
|---|---|---|---|---|
| BBR | 1.00 (1.00) | 1.00 (1.00) | 1.00 (1.00) | 1.00 (1.00) |
| CUBIC | 0.99 (0.94) | 0.99 (0.97) | 0.99 (0.96) | 1.00 (0.98) |
| RENO | 0.99 (0.98) | 0.99 (0.95) | 0.99 (0.96) | 1.00 (0.99) |
| CUBIC-p | 0.99 (0.95) | 1.00 (0.97) | 0.99 (0.96) | 1.00 (0.99) |
| RENO-p | 1.00 (0.98) | 0.99 (0.96) | 0.99 (0.97) | 1.00 (0.99) |
| | Overall Accuracy: 0.99 (0.97) | | | |

**Table 4: Identification metrics for multi-host network before (after) 50Mbps bottleneck.**

| | Precision | Recall | F1 | Accuracy |
|---|---|---|---|---|
| BBR | 1.00 (1.00) | 1.00 (1.00) | 1.00 (1.00) | 1.00 (1.00) |
| CUBIC | 0.98 (0.97) | 0.88 (0.92) | 0.93 (0.94) | 0.97 (0.98) |
| RENO | 0.89 (0.92) | 0.98 (0.97) | 0.93 (0.95) | 0.97 (0.98) |
| CUBIC-p | 0.97 (0.96) | 0.93 (0.89) | 0.95 (0.93) | 0.98 (0.97) |
| RENO-p | 0.93 (0.90) | 0.97 (0.96) | 0.95 (0.93) | 0.98 (0.97) |
| | Overall Accuracy: 0.95 (0.95) | | | |

**Table 5: Identification metrics for cross-traffic network before (after) shared link**

therefore convinced that our approach's susceptibility to bufferbloat alleviates with concurrent flows. We attribute this to the stronger influence of the cwnd onto the flow rate when multiple flows compete as we described before in Section 4.2.1.

*4.2.4 Arbitrary Vantage Points.* Until now, we have investigated what happens if packet arrivals are captured at the edge - either before or after a single link. However, in reality, a flow will traverse multiple links, so there are multiple vantage points to observe the flow. Moreover, there might cross flows which have been shaped by different bottlenecks interacting with the flow to be observed.

Thus, we now focus on what happens if we measure at a vantage point in the middle of a network, where flows with different bottlenecks compete. To evaluate this scenario, we use our initially trained model on the cross-traffic network.

The performance results can be seen in Table 5 and are surprisingly similar to the multi-host network after the bottleneck. We achieve an overall accuracy of 95% which indicates that our approach generalizes also to arbitrary vantage points although not trained for with remarkable performance regarding distinguishing between BBR and loss-based CC with an F1 score of 100%.

*4.2.5 Required Flow Duration.* The previous results were obtained with flows of 60 seconds length but flows this long occur relatively seldom [25]. Therefore, we now evaluate the flow duration needed for correct classification over the single-host and multi-host network vantage points at 50Mbps. The F1 scores depending on the flow duration sampled in 512ms steps are shown in Figure 4. Generally speaking, we can see that the identification performance increases when flows can be observed over a longer time.

BBR (due to its characteristic behavior in and after slowstart) can be identified with an F1 score of 85% in 1024ms in the multi-host setting / in 3584ms in the single-host setting. The unpaced variants need 4096ms to achieve an F1 score over 75%, while the paced variants need 2048ms to achieve the same in the multi-host setting. In the single-host setting, 75% are achieved in 8706ms for

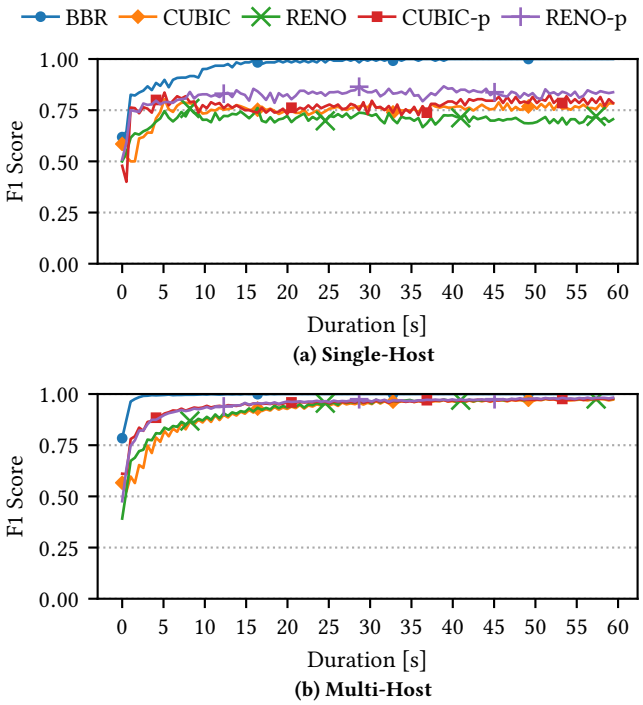



**Figure 4: F1 score w.r.t. flow duration for single and multi-host network. The duration is increased in 512ms steps. The bottleneck was limited to 50Mbps and the delays of the multi-host network are used for comparability.**

the unpaced variants and 1536ms for the paced variants. However, the scores are very unsteady and fluctuate such that, e.g., RENO's F1 score approaches 75% only for certain moments.

We also found a trivial correlation for loss-based CC between delay and classification duration (longer delay $\hat{=}$ longer duration).

## 5 CONCLUSION

In this paper, we presented the design and performance of DeePCCI, a passive method to identify congestion control variants using just packet arrival time information. DeePCCI does not need header information, visibility into the reverse path nor domain knowledge during training of the variants that it wants to identify. It is thus designed for a QUIC-enabled world and can be easily adapted for new congestion control variants. Our results indicate a promising performance over a diverse set of vantage points. Especially variants which eminently vary in behavior in comparison to traditional approaches, such as BBR, can be distinguished with remarkable performance throughout most tested scenarios. Yet, the approach is challenged in distinguishing loss-based CC when it cannot find enough characteristic features, e.g., when only one flow sends data.

We believe that this performance can be partly increased when training and optimizing for specific network properties, i.e., when knowing where DeePCCI should be used and how many flows compete, by, e.g., adapting the histogram bin-size. In the future, we plan to evaluate DeePCCI on real-world network traces, on further congestion control variants, when subject to AQMs affecting the identification abilities as well as how receiver ACK generation or ACK compression affect the forward path and thus DeePCCI.

## ACKNOWLEDGMENTS

We would like to thank Stefan Koltermann for initially providing us with GPUs. Further computing resources were granted by RWTH Aachen University under project thes0601. This work has been funded by the DFG as part of the CRC 1053 MAKI and within the Cluster of Excellence "Internet of Production" (IoP) under project ID 390621612.

## REFERENCES

[1] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, 5 (2016).
[2] G. Casagrande, F. Granelli, and D. Miorandi. 2011. TCPMoon: Monitoring the Diffusion of TCP Congestion Control Variants in the Internet. In *IEEE ICC*.
[3] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *USENIX NSDI*.
[4] J. Gettys and K. Nichols. 2011. Bufferbloat: Dark Buffers in the Internet. *ACM Queue* 9, 11 (2011).
[5] S. Ha, I. Rhee, and L. Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *ACM SIGOPS Oper. Syst. Rev.* 42, 5 (2008).
[6] D. H. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure. 2018. Recurrent Neural Network-Based Prediction of TCP Transmission States from Passive Measurements. In *IEEE NCA*.
[7] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE CVPR*.
[8] S. Hochreiter and J. Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997).
[9] M. Hock, R. Bless, and M. Zitterbart. 2017. Experimental Evaluation of BBR Congestion Control. In *IEEE ICNP*.
[10] S. Ioffe and C. Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*.
[11] J. Iyengar and M. Thomson. 2019. *QUIC: A UDP-Based Multiplexed and Secure Transport.* Internet-Draft draft-ietf-quic-transport-18.
[12] V. Jacobson. 1988. Congestion Avoidance and Control. In *ACM SIGCOMM*.
[13] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. 2004. Inferring TCP Connection Characteristics Through Passive Measurements. In *IEEE INFOCOM*.
[14] D. P. Kingma and J. Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* (2014). http://arxiv.org/abs/1412.6980
[15] B. Lantz, B. Heller, and N. McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *ACM HOTNETS*.
[16] Y. LeCun and Y. Bengio. 1998. The Handbook of Brain Theory and Neural Networks. Chapter Convolutional Networks for Images, Speech, and Time Series.
[17] J. Nagle. 1984. *Congestion Control in IP/TCP Internetworks.* RFC 896.
[18] J. Oshio, S. Ata, and I. Oka. 2009. Identification of Different TCP Versions Based on Cluster Analysis. In *IEEE ICCCN*.
[19] J. Padhye and S. Floyd. 2001. On Inferring TCP Behavior. In *ACM SIGCOMM*.
[20] V. Paxson. 1997. Automated Packet Trace Analysis of TCP Implementations. In *ACM SIGCOMM*.
[21] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld. 2018. A First Look at QUIC in the Wild. In *PAM*.
[22] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak. 2015. Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks. In *IEEE ICASSP*.
[23] K. Simonyan and A. Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014).
[24] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu. 2014. TCP Congestion Avoidance Algorithm Identification. *IEEE/ACM Transactions on Networking* 22, 4 (2014).
[25] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. 2002. On the Characteristics and Origins of Internet Flow Rates. In *ACM SIGCOMM*.