How Attention Deep Learning Can Improve Copa Congestion Control Performance

Victor Perrier
ISAE-SUPAERO, TéSA
Toulouse, France
victor.perrier@isae-supaero.fr

Emmanuel Lochin
ENAC, TéSA
Toulouse, France
emmanuel.lochin@enac.fr

Jean-Yves Tourneret
ENSEEIHT, IRIT, TéSA
Toulouse, France
jean-yves.tourneret@enseeiht.fr

Nicolas Kuhn Patrick Gelard
CNES
Toulouse, France
firstname.lastname@cnes.fr

***Abstract*—Most modern congestion control algorithms, that aim to optimize delay and throughput, exploit more metrics than the sole packet loss congestion information. These additional metrics are mostly based on the round trip time evolution and allow congestion controls to reach better performance, in particular on wireless and cellular links as demonstrated by Copa, BBR, or REMY. Basically, these metrics allow congestion control to estimate the queuing level of the path and its evolution, to assess the presence of congestion. Actually, a good estimation of this level obviously prevents congestion losses, but also allows assessing a ratio of error link losses among the whole observed losses. The consistency and accuracy of these metrics are key to good congestion control performance, and this explains, for instance, the good performance of Copa currently in production at Facebook. However, these metrics remain challenging and the quest of an accurate and practical estimation seems complex. This paper investigates how a novel deep learning algorithm, known as Attention, can help in assessing queuing evolution and status on an end-to-end path. Among others, we focus on the evolution of the total time spent by packets in the buffers, which is the key metric of Copa. The results unequivocally demonstrate a better accuracy of this metric used by Copa.**

***Index Terms*—TCP, Copa, BBR, Congestion Control**

I. Introduction

New congestion control (CC) mechanisms such as Copa [1], REMY [2], or BBR [3] have demonstrated better performance [4] than CUBIC, the historical TCP Newreno or any TCP variant. CUBIC is the current congestion control deployed over GNU/Linux for instance, BBR is currently deployed by Google, and Copa is a congestion control for improved video performance deployed by Facebook[1]. Such performance can be achieved due to the addition of novel metrics compared to the standard ones based on loss signal and timeout estimation.

The algorithm REMY [2], based on machine learning (ML), was a pioneer in this domain. However, as explained in [4] and [5], the pace of convergence to achieve a consistent CC algorithm can take more than 24 hours offline learning and remains specific to a given architecture. Following REMY, some improvements based on a similar ML approach have been proposed with Verus [6], PCC [7], PCC-Vivace. Copa and BBR rethink the way CC should perform by determining an optimal operating regime. The former is based on control theory while the latter is closer to a standard CC algorithm; and both of them introduce new sensing metrics to assess the network congestion level. This optimal objective is given by a theoretical point corresponding to the standard TCP knee phase [8].

As a matter of fact, all these novel CC strategies lay on novel metrics such as total queuing delay as for instance, in [1], round trip time (RTT) or minimum RTT in [3], [1], bottleneck bandwidth in [3], etc. While these CC focus on bringing out new congestion management, the accuracy of the metrics estimated greatly influences their performance. This is the rationale of this paper which investigates new metrics of interest and improves the estimation of existing ones. Our objective is not to propose a novel CC candidate or seek to compete with the numerous existing proposals but to improve the accuracy of these novel metrics proposed by these new CC using recent trends in ML.

The paper is structured as follows: the next section presents the basics of new CC algorithms. We then introduce and explain the metrics of interest; afterward we present the experimental setup used to validate our assumptions; later this paper focuses on explaining the choice of deep learning algorithms and how we trained them. Finally, the last part presents and explains our results.

II. In quest of the optimal control point

The idea developed in this paper follows a common trend among recently proposed CC mechanisms: instead of using only loss signals of a packet to assess the network congestion level (meaning that no acknowledgment packet has been received for a given sent packet, or a timeout has been triggered), we define new metrics to better estimate the network congestion level. As explained in [3] that adopts a hybrid approach combining delay and loss signals, congestion control should remain at the optimal control point (see Fig. 1), achieving full capacity while minimizing the queue load. However, to reach this optimal point, we need better insight into the queuing evolution of the network bottleneck rather than considering simple loss signals as illustrated in [3].

To better assess what is this optimal control point, let's consider a simple model where a fixed-sized bottleneck queue with a service μ bit/s, a max queue size q_m, and a current queue size or load q_l, is crossed by a single flow at λ bit/s. This flow might encounter three states: 1) if $\mu > \lambda$, the queue is always empty and packets are passed without delay; 2) if

[1]https://engineering.fb.com/2019/11/17/video-engineering/copa/

$\mu < \lambda$ and $q_l < q_m$, packets are then stored and the end-to-end delay increases; 3) finally if $\mu < \lambda$ and $q_l = q_m$, arriving packets are dropped and severe congestion occurs. The delay and delivery characteristics considering this bottleneck link are illustrated in Fig. 1. The problem is thus to operate as close as possible to this optimum operating point. However and this is the rationale of this paper, this optimal point cannot be reached without an accurate estimation of the network congestion level.

TCP Vegas was the first attempt to solve this problem based on a delay-based CC that estimates the RTT of a flow by tracking the time of a sent packet and its corresponding received ACK (acknowledgment packet). The objective was to interpret a raise of the RTT as a congestion signal, as illustrated by the state 2) above (i.e. $\mu < \lambda$ and $q_l < q_m$). However, TCP Vegas has poor performance in several scenarios [9], which has led to a hybrid approach combining delay and loss-based approaches. This was at the origin of BBR, a TCP delay-based algorithm [3]. BBR is quite similar to TCP Vegas but attempts to determine both the bottleneck capacity of the path and the RTT. In other words, BBR combines a novel metric to improve the robustness of the CC algorithm.

Copa and BBR are the two main CC mechanisms that aim at operating close to this optimal point. Both algorithms use delay signals to detect congestion and sense the network load with a flickering of the capacity as illustrated with Copa in Fig. 2. When they are in the congestion avoidance phase, both mechanisms are trying to increase and reduce the throughput periodically to probe the network congestion level. However, as shown in [3], these metrics are influenced by several factors and are difficult to estimate. Once again, this motivates the present study, which is to explore novel metrics and schemes to correctly estimate them.

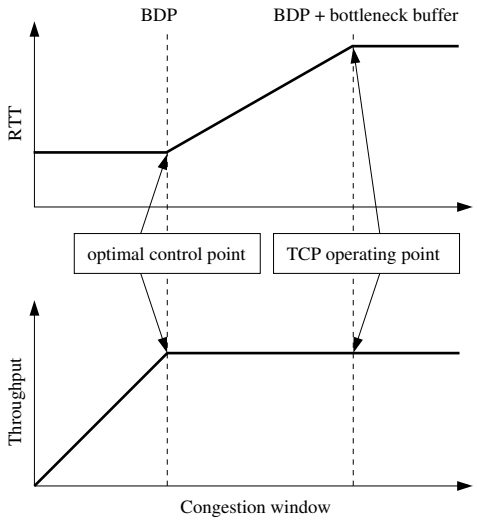


Fig. 1: Optimal control point as shown in [3].

III. Metrics of Interest

All CC strategies detailed in the previous section basically attempt to estimate the bottleneck queuing size or evolution.

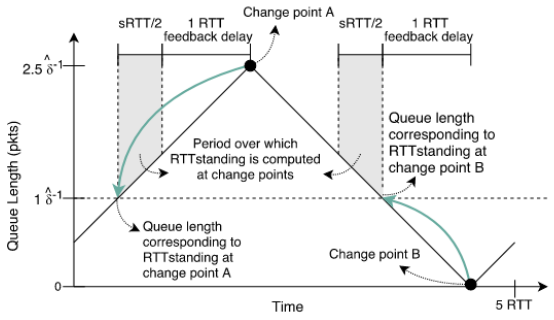


Fig. 2: Copa mechanism following [1].

The optimal regime targeted by BBR, Copa, and other new algorithms is thus strictly linked to the queuing evolution of the bottleneck. As a matter of fact, we propose to investigate three new metrics of interest used jointly with an existing metric, and to assess their relevance and accuracy.

These new metrics are introduced below and illustrated in Fig. 3:

- Y_1 is the bottleneck buffer size. Note that the bottleneck buffer size (with respect to the maximum size) corresponds to the maximum size of all the buffer sizes along the path. The heuristic behind that metric is that there is usually one bottleneck limiting the global performance;
- Y_2 is the slope of the bottleneck buffer size over the last N packets. This feature is computed using a regression over the available data (if the packet is lost, we cannot access the buffer load, because the packet never came through). This feature is important since knowing if a buffer is being filled or emptied is critical for a CC algorithm. This knowledge can help the optimal point to be stabilized, (1);
- Y_3 is the average time spent inside buffers over the last N packets. This feature is similar to Y_1, except that all the queues that are on the path are considered. Note that this metric is also used in the Copa mechanism;
- Y_4 is the slope of the time spent inside buffers over the last N packets. This feature is the equivalent of Y_2 for the variable Y_3.

In an obvious manner, having an accurate estimation of these metrics will undoubtedly improve the performance of a congestion control algorithm.

IV. Experimental Setup and Scenario

The experimentations were conducted using the Mininet emulator and validated using the parking lot scenario depicted in Fig. 4. Each link has a capacity of $50\,Mbps$, and a nominal delay of $10\,ms$. Each queue enables FQ-CoDel by default with a size of $500\,pkts$. Packets are sent from the client to the server. TCP flows are sent between c1, c2, c3, and c4 to maintain a constant network load. Several studies show that short-lived flows, mainly generated by Web data transfers caused by user interactions, dominate the Internet traffic [10]. Thus, the TCP traffic is generated in such a way that the length of the TCP flows respects the Pareto principle (80% of short

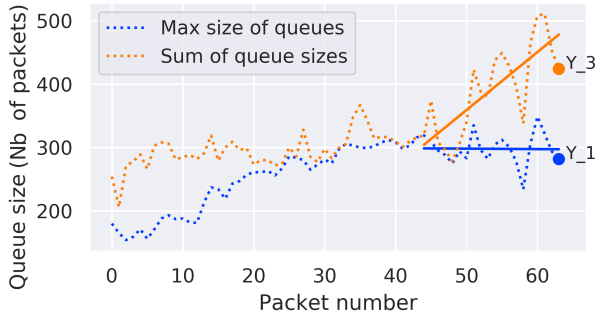


Fig. 3: Visualisation of the metrics. The two plots correspond to the sum of buffer sizes, and to the maximum of the buffer size along the path of a packet. Y_1 is the maximum of the buffer size for the last packet. Y_2 is the slope of the plain blue line. Y_3 is the sum of all buffer sizes during the last packet transmission. Y_4 is the slope of the plain orange line.

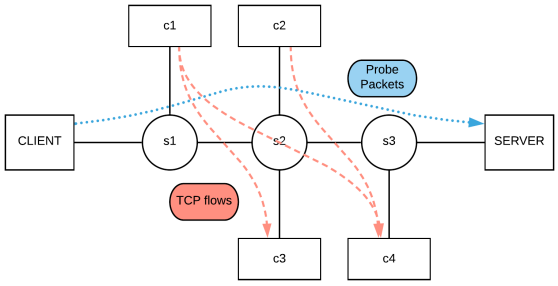


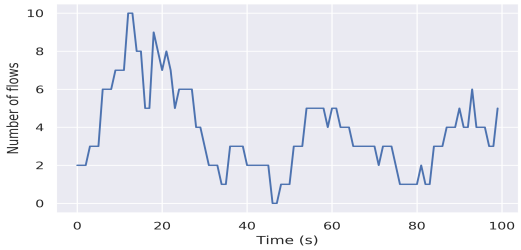
Fig. 4: Parking lot topology used for the tests.

flows, 20% of long flows) as shown in Fig. 5 over the long run. The objective of this first experimentation is to probe the network congestion level, i.e., the load of the queues at s1, s2, and s3. The probe flow follows the blue path outlined in Fig. 4 and is a TCP flow containing real data. Each TCP flow following the red path has been generated as follows:

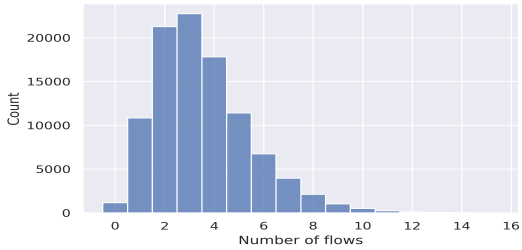
- the duration of each TCP connection has a Poisson distribution with parameter $\lambda = 1$;
- the time between two starting flows has an exponential distribution with parameter $\lambda = 10$;
- the server-client pair is randomly selected between the pairs (c1, c3), (c1, c4) and (c2, c4);
- each TCP flow is generated with the iPerf traffic generator.

This setup enables realistic and variable network conditions:

- the bottleneck is not stable and moves from s1 to s2 randomly. We can see the evolution of the queue length in Fig. 6: both queues can act as the bottleneck depending on the network load and flows;
- the number of TCP flows changes to mimic a network load as shown in Fig. 5;
- TCP constantly switches between the slow-start phase and the congestion avoidance phase. This allows us to obtain a very diverse distribution of the variables, representing various kinds of behavior (few flows, congestion, slow-start/cruise-control phase, unbalance between buffer load...).



(a) Evolution of the number of flows in the network for 100 seconds. The alternation between high load phases and phase with almost no TCP flows can be observed.



(b) Histogram of the number of concurrent flows in the network with a 1 second resolution. There is a majority of short flows.

Fig. 5: Visualization of some flow characteristics.

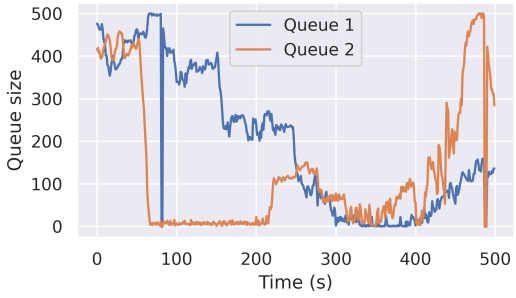
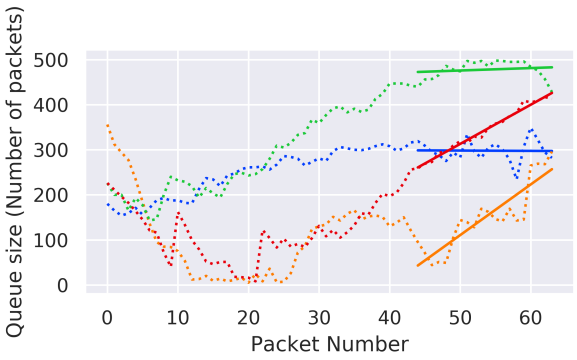


Fig. 6: Example of the evolution of the two bottleneck queue lengths at $s1$ and $s2$ for two different time periods.

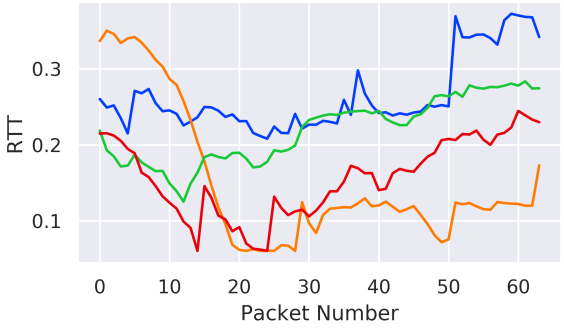
V. Machine Learning (ML) algorithms used in this study

This section investigates the use of ML algorithms to estimate the aforementioned metrics. We seek to analyze whether we can correlate the network congestion level (i.e., the fours metrics introduced earlier) with the data that CC algorithms have access to (ie, delays, loss signals, sending rate/window, timestamp of sent packets). Note that this is a blind spot of recent CC mechanisms, which have introduced simple models to estimate some of the network internal variables (e.g., the time spent inside the buffers).

To illustrate that, we would like to estimate what is happening in 7b with the data from 7a. In some cases, a correlation can be guessed, but an accurate model needs to be constructed.



(a) RTT.



(b) Bottleneck queue size.

Fig. 7: Examples of evolution of some metrics during a window of 64 consecutive packets.

A. *Choice of the prediction algorithm.*

In this section, we introduce the different ML algorithms used is this study and justify both their choice and the validity of the results obtained with these algorithms. We have an input of dimension $3N + 1$ containing both discrete (Boolean) and continuous variables. The output that we seek to predict is one of the continuous metrics, which restricts the type of algorithm that can be considered. We also need a predictor adapted to regression.

Our first guesses were the use of feed-forward neural networks, support vector regressions (SVRs), and traditional time series prediction (ARIMA for example). Feed Forward Neural networks are quite simple to use but can be hard to tune, because of a high number of hyper-parameters (number of layers, size of the layers, choice of the activation function).

However these kinds of algorithms are not adapted to our problem for the following reasons :

- the input size is fixed for feed-forward neural networks and SVRs. If these algorithms are trained with a pattern size of 64, it is impossible to extend the results to samples having a different size. We would like to have an algorithm that can adapt to the length of the time-series. Indeed, the available amount of data depends on the capacity of the link (the faster the link, the more data), and of course, including all the data might improve the accuracy of the results;
- after consulting the state-of-the-art, we noticed that Copa for example uses a simple estimator in order to compute Y_3, based on $RTT_{standing} - RTT_{min}$. $RTT_{standing}$ is the minimum of the RTT for a short time window whereas RTT_{min} is the minimum for a longer time period. Copa works quite well, so our idea is that being able to compute a minimum from a vector is certainly important if we seek to get a good estimation of some metrics. However, it is very difficult for the mentioned ML algorithms to estimate a maximum function (it is theoretically possible with neural networks with large complexity but is difficult in practice).

In order to fill these gaps, we have studied existing ML algorithms that are adapted to time series. These algorithms include the family of long short-term memory (LSTM) networks [11] and GRU [12]. While these algorithms allow the first concern to be bypassed, they still do not allow a minimum/maximum to be computed. We also investigated Attention [13]. Attention has become state-of-the-art solution for natural language processing, and is adapted to time series as well. We found that Attention allows us to estimate the maximum/minimum of a vector with ease. Indeed, Attention mechanism involves the following computation: $Y = softmax(\frac{QK^T}{\sqrt{n}})$ with Q K and V linear transformations of our input vector X, and n the length of the vector. To illustrate that, if we choose the simplest linear transformation $X = V = Q = K$, the resulting vector is an estimation of the maximum of X where X could be a vector of RTT expressed in *ms*.

For instance $X = [9.3246, 10.4722, 11.5280, 12.6615, 10.6212]$

leads to

$$Y = [12.6147, 12.6317, 12.6417, 12.6486, 12.6334]$$

Of course, Attention is more complicated than this example, and knowing the maximum/minimum is not enough to make a correct prediction. However, this example illustrates the intuition which lead us to consider that Attention should be adapted for our task. It was indeed confirmed by the results: for the moment, Attention-based algorithms achieve the best performance on our datasets. We thus have chosen to use such a scheme.

B. *Features*

As mentioned before, we seek to use ML algorithms to predict the network congestion level. As we are using an emulated environment, the internal congestion level is known at each time instant, which allows supervised ML algorithms to be considered. We use then the data collected by the probe flow to estimate the metrics. These collected data are:

- the average rate at which the packets are sent;
- a binary variable indicating which packets have been lost;
- the RTT of each packet, as shown in Fig. 7a. Note that we have used the complete time series in this work;
- a timestamp indicating when each packet was sent.

As a consequence, the vector of features that will be used at the input of the ML algorithm is of size $3N + 1$.

C. Robustness

The main issue with ML algorithms is that they will not generalize if learning is conducted in a restricted learning space (i.e., our emulated environment), which is different from the real world. In order to study this generalization capability from our algorithm, we trained our algorithm in a restricted environment, and then tested it with different parameters. This way, it was possible to determine whether the proposed predictor is robust to changes in some of the network parameters. If some parameter changes have an impact on the prediction, the learning data set needs to be improved, and include a higher diversity to improve generalization. The parameters that we changed to test the robustness of our estimator include:

- the topology of the network: the number of nodes between the client and the server;
- the capacity of the links;
- the delay of the links;
- the queuing/scheduling mechanism in the buffers.

VI. Results

Neural networks were trained in a supervised way to predict the internal congestion level from the features provided by the CC algorithm. The predictions of our algorithm are then evaluated in two different ways:

a) error plots: the x axis represents the real value of the variable to be predicted whereas the y axis is the average error resulting from that prediction. The line represents the average of the estimation error, and the area around the line shows the standard deviation, indicating how the error is spread around its average value. Note that the y axis has been normalized by the maximum range of the x axis, to allow comparison within the total range;

b) multivariate distributions: the actual value of the variable is represented in the x-axis whereas its prediction is in the y-axis. This representation shows how the predictions are distributed around their means. Note that between two consecutive blue lines, there are 10% of the points of the scatter plot. The top panel of each figure shows the distribution of the variable we try to predict (Y_1, Y_2, Y_3 or Y_4). Finally, the right panel shows the distribution of the predicted variable.

Fig. 8a shows that the bottleneck load can be predicted with good accuracy with the proposed ML algorithm. The important point is to know if the bottleneck is full or in an empty state, in order to work close to the optimal point corresponding to the queue almost empty, but used at 100% of its capacities. The prediction of Y_2 (Fig. 8b) is correct and it provides important information, namely if the queue is being filled or emptied. However, the prediction around the edge is slightly worse: the evolution speed is always underestimated. Table I shows performance measures for the prediction of the Boolean variable $(Y_2 > 0)$. We can observe that the prediction is correct in 84% of cases. We think that this information is also very useful for a CC algorithm, and is complementary to the knowledge of Y_1 and Y_3 for finer control. For example, if we are already working at the optimum control point, and $(Y_2 > 0)$, then we can slightly reduce the sending window to stabilize Y_2 around zero, with a low value of Y_1.

The proposed ML algorithm is also predicting Y_3 (Fig. 8c) with high accuracy, when compared to the simple Copa predictor ($RTT_{standing} - RTT_{min}$): both methods underestimate the total load on all the buffers (Y_3), but our approximation error is approximately half of that obtained with Copa. Note again that our aim is not a comparison with Copa or BBR, but rather to improve the performance of existing CC algorithms. Predictions of Y_1 and Y_3 show that the proposed algorithm tends to overestimate the load when the queue is almost empty, and to underestimate the load when it is not empty. The estimation provided by Copa is sensitive to the same phenomenon.

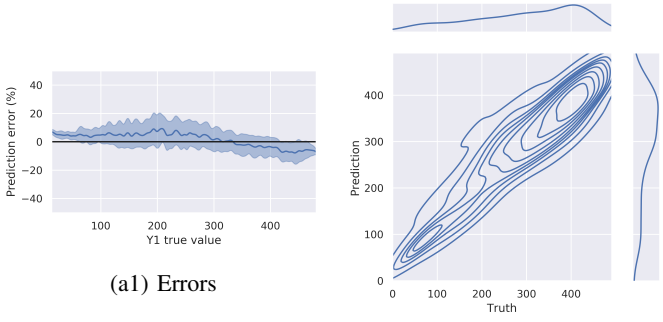
This can be explained by the following reasons:

- when the buffers are full, Copa estimates the load proportionally to $RTT_{standing} - RTT_{min}$. This estimation works if, during the aforementioned time window, the buffers are empty. This way RTT_{min} should correspond to the RTT caused by the link, and the estimation should be correct. However, in practice, especially when we work with large buffers, the queues are not emptied fully, and $RTT_{standing}$ is just an overestimation of the RTT of the link. That leads to an underestimation of the buffer load. Copa uses this estimation with the hypothesis that Copa is the only CC being used, but this scenario is not realistic: in a lot of cases, there are different CC algorithms in competition;
- our ML algorithms based on neural networks or SVRs are black boxes. Thus, we cannot explain how the load is precisely computed. Note that a similar underestimation is observed for these algorithms and for Copa. We think that this can be explained as follows: if we are working on a big buffer, we will rarely see the empty state of that buffer, since the algorithm cannot differentiate between the RTT caused by the state of the link and the RTT caused by the minimal load in the buffer;
- there is also a small overestimation of the load of the buffer

TABLE I: Scores for Y_2, the slope the bottleneck buffer size over the last 15 sent packets.

	Results
True positives	36 %
False positives	8%
False negatives	8%
True negatives	47%
Precision	0.81
Recall	0.81

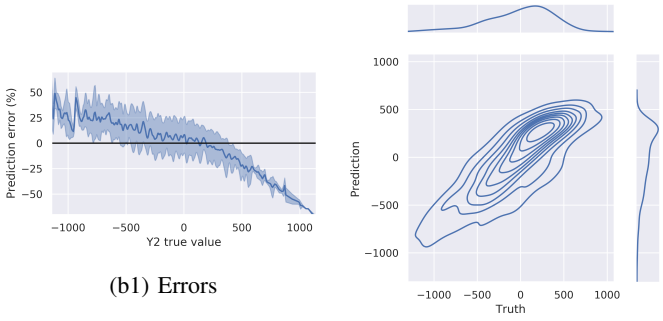
when the queues are almost empty.



(a1) Errors

(a2) Distribution

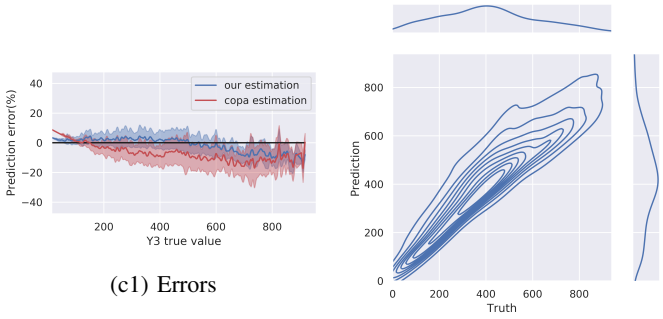
(a) Y_1: average size in packets of the bottleneck, for the last 15 sent packets.



(b1) Errors

(b2) Distribution

(b) Y_2: slope of the average size in packets of the bottleneck, for the last 15 sent packets.



(c1) Errors

(c2) Distribution

(c) Y_3: average size in packets of all the buffers along the path of the packets, for the last 15 sent packets.

Fig. 8: Error and prediction of the three variables using Attention (the error is normalized with the maximum range of the variable).

VII. Conclusion

This paper introduces metrics of interest that aims to increase CC algorithms overall performance. These metrics are : the buffer size at the bottleneck; the evolution trend of that buffer; the total delay on the current path due to network congestion; and finally the evolution trend of that delay. Estimating these metrics with good accuracy allows CC algorithms to better perform. Indeed, these metrics provide a consistent estimation of the network congestion level, and allow CC algorithms to correctly react to congestion. We propose a prediction method based on Attention allowing challenging metrics to be estimated accurately. To validate our proposal, we perform real measurements with Mininet. All the measurements have been challenged by changing the network topology and the environment variables to effectively demonstrate the robustness of the method proposed.

Determining whether the pacing of consecutive packets can lead to a better prediction of the aforementioned metrics is an interesting prospect: as shown in [14], if the packets are sent following a specified pattern, some information can be gained, especially if we are already working on metrics based on delay. As a result, we are currently investigating the impact of specific patterns on the accuracy of the metrics estimated.

VIII. Acknowledgments

The authors would like to thank ISAE-SUPAERO and CNES for their funding support.

References

[1] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *USENIX NSDI*, Renton, WA, Apr. 2018, pp. 329–342.

[2] K. Winstein and H. Balakrishnan, "TCP ex machina: computer-generated congestion control," in *ACM SIGCOMM Conference*. Hong Kong: ACM, Aug. 2013, pp. 123–134.

[3] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Commun. ACM*, vol. 60, no. 2, p. 58–66, Jan. 2017.

[4] F. Y. Yan *et al.*, "Pantheon: the training ground for internet congestion-control research," in *USENIX ATC*, Boston, MA, Jul. 2018, pp. 731–743.

[5] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, p. 123–134, Aug. 2013.

[6] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 509–522, Aug. 2015.

[7] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *USENIX NSDI*, Oakland, CA, May 2015, pp. 395–408.

[8] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, 1989.

[9] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation of tcp vegas: Emulation and experiment," in *ACM SIGCOMM*, New York, NY, USA, 1995, p. 185–195.

[10] D. Ciullo, M. Mellia, and M. Meo, "Two schemes to reduce latency in short lived TCP flows," *IEEE Communications Letters*, vol. 13, no. 10, Oct. 2009.

[11] S. Hochreiter and J. Schmidhuber, "LSTM can solve hard long time lag problems," *Advances in neural information processing systems*, pp. 473–479, 1997.

[12] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37. Lille, France: PMLR, Jul. 2015, pp. 2067–2075.

[13] A. Vaswani *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[14] V. Konda and J. Kaur, "Rapid: Shrinking the congestion-control timescale," in *IEEE INFOCOM*, Rio de Janeiro, Brazil, 2009, pp. 1–9.