Learning Congestion State For mmWave Channels

Talal Ahmad
New York University
ahmad@cs.nyu.edu

Shiva Iyer
New York University
shiva.iyer@cs.nyu.edu

Luis Diez
University of Cantabria
ldiez@tlmat.unican.es

Yasir Zaki
New York University Abu Dhabi
yasir.zaki@nyu.edu

Ramón Agüero
University of Cantabria
ramon@tlmat.unican.es

Lakshminarayanan Subramanian
New York University
lakshmi@cs.nyu.edu

ABSTRACT

Millimeter wave (commonly known as mmWave) is enabling the next generation of last-hop communications for mobile devices. But these technologies cannot reach their full potential because existing congestion control schemes at the transport layer perform sub-optimally over mmWave links. In this paper, we show how existing congestion control schemes perform sub-optimally in such channels. Then, we propose that we can learn early congestion signals by using end-to-end measurements at the sender and receiver. We believe that these learned measurements can help build a better congestion control scheme. We show that we can learn Explicit Congestion Notification (ECN) per packet with an F1-score as high as 97%. We achieve this by doing unsupervised clustering using data obtained from sending periodic bursts of probe packets over emulated 60 GHz links (based on real-world WiGig measurements), with random background traffic. We also describe how the learned values of ECN can be utilized for rate estimation.

KEYWORDS

mmWave, machine learning, congestion control, ECN

ACM Reference Format:
Talal Ahmad, Shiva Iyer, Luis Diez, Yasir Zaki, Ramón Agüero, and Lakshminarayanan Subramanian. 2019. Learning Congestion State For mmWave Channels. In *3rd ACM Workshop on Millimeter-wave Networks and Sensing Systems (mmNets'19), October 25, 2019, Los Cabos, Mexico.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3349624.3356769

1 INTRODUCTION

2 BACKGROUND AND RELATED WORK

The millimeter wave bands have the potential of enabling high throughput communication with the additional caveat of rapidly fluctuating links over few wavelengths. Due to the high carrier frequency, mmWave communications suffer from huge propagation loss, reducing the coverage area of base stations. In addition to this, due to weak diffraction ability mmWave communications are sensitive to blockage by obstacles such as humans and furniture [11]. Therefore, line of sight and non-line of sight communication can experience significantly different channel throughput.

TCP variants are known to suffer from a number of inefficiencies in these kinds of scenarios. For example, TCP Cubic suffers from the bufferbloat problem due to large window sizes, which results in extremely high packet delays in cellular networks [15, 17]. Another issue that TCP fails in addressing in cellular networks are packet losses that are not linked to congestion; Hu et. al. [8] identified packet reordering as a common occurrence in cellular networks and being mistaken as a congestion related loss.

Shortcomings of legacy TCP over highly varying cellular channels have led to newer delay-based congestion control protocols like Sprout [15] and Verus [17] that were specifically designed for cellular. While Sprout focuses on the problem of reducing self-inflicted queuing delays and uses packet inter-arrival times to detect congestion, Verus was designed to create a balance between packet delay and throughput. Recently, PCC Vivace [5] has been shown to react well to changing networks while alleviating the bufferbloat problem. PCC Vivace leverages ideas from online (convex) optimization in machine learning to do rate control.

Congestion control has been shown to improve with explicit feedback from network, the examples of this feedback can range from available bandwidth at a time to Explicit Congestion Notification (commonly known as ECN) marker [1, 4, 9]. These protocols yield great result but require specific in network hardware to provide feedback to the sender. Therefore, it has been difficult to deploy them at larger scales. In our approach, we leverage un-supervised learning to learn this ECN feedback from network and give a sketch of how it can be used to develop a congestion control algorithm. There has been other work on learnability of congestion control [14] that are more general, while we specifically try to learn an ECN signal. There has also been prior work on using machine learning to improve congestion control for wireless channels [7]. Comparing our approach against such similar works will be part of future work.

3 CONGESTION CONTROL OVER 5G

We explored the performance of two different algorithms, Verus [17] and Sprout [15], using a setup similar to Pantheon [16] local tests. We chose these two as they both are designed for highly varying cellular channels and react quickly to changing channel conditions. We used a Mahimahi [10] linkshell to emulate an mmWave link from a trace file that was generated using a real WiGig router leveraging the approach shown in §5.1. The underlying conditions

mmNets'19, October 25, 2019, Los Cabos, Mexico

ACM ISBN 978-1-4503-6932-9/19/10…$15.00
https://doi.org/10.1145/3349624.3356769

used to produce the highly varying trace were random movements of humans between a WiGig sender and a receiver radio. The trace used was 60 seconds long as we wanted to emulate a long running data flow. There was no background traffic so as to see the full potential of these algorithms.

Figure **??** shows the performance of these algorithms in terms of achieved throughput and one-way queuing delay experienced as we ran a single long running flow. The average capacity of the trace was 51.6 Mbps. The queue used in these experiments was a standard droptail queue and the size was set to the bandwidth delay product (BDP). To calculate the BDP, we used the maximum observed capacity value of the link since the link is highly varying. We set the link propagation delay to 20 ms using the Mahimahi delayshell.

Having similar design goals, both of them succeed in keeping the average delay reasonably low, at 84 ms for Sprout and 191 ms for Verus. However, neither is able to keep up with the highly varying channel in terms of throughput, and consequently the channel ends up being grossly underutilized in both cases. Sprout is able to maintain an average throughput of only 11.3 Mbps (21% utilization), while Verus does slightly better at 17.1 Mbps (33% utilization). Sprout is not agressive in ramping up because it tries to keep the delay low. Verus is more aggressive than Sprout but is still not aggressive enough for mmWave links.

While delay-based algorithms like Verus and Sprout have been shown to work on certain highly varying links, clearly they have their limitations, and those limitations become even more stark at the range of mmWave, where channel fluctuations are extremely rapid. Thus we realize that we need more than a purely delay-based approach to congestion control in mmWave scenarios. In the past, other types of feedback, such as an Explicit Congestion Notification (ECN) marking in the network have been used to devise improved congestion control mechanisms[1]. However, since such signals require specialized hardware, we explore the ability to predict such early signals of congestion using a purely data-driven approach, from end-to-end delay and packet arrival time information observed at the sender.

4 LEARNING ECN

We showed in the last section that end-to-end delay feedback takes us only so far in the case of highly fluctuating network conditions, and thus we need a different kind of feedback from the underlying network. In-network congestion feedback like an Explicit Congestion Notification (ECN) [12] can deliver extra information with each packet received and has been shown to improve performance of congestion control algorithms in specialized network environments [1]. ECN uses two bits in the IP header to mark congestion in ECN-enabled routers. The high-level idea is that ECN-enabled routers can set the Congestion Experienced (CE) codepoint (this is when both the ECN bits have a value of 1) in the IP header of packets instead of doing active queue management. For example, a Random Early Detection (RED) queue [6] marks the CE codepoint in packets with a probability p if the queue size increases beyond a threshold. In case of DCTCP, CE codepoint is marked by a router if the queue reaches certain threshold.

We propose to learn the ECN value for each packet using an unsupervised learning approach. For our setup we assume the ECN is marked 1 with a probability of 100% if the router buffer is filled more than 50%. So all of the packets correlating with that buffer state are assumed to have ECN 1.

We have a simple setup similar to § 3 where we have a sender and receiver communicating with each other over a WiGig channel. The sender wishes to send data at the maximum rate possible without causing congestion in the network. In addition, there exist some background traffic in the link, which is unknown to the sender. We send probe bursts of packets into the network and try to use the measurements from these probes to predict the value of ECN for each packet. In particular we use the RTT and inter-arrival time (IAT) of each packet to do the prediction. While the RTT is readily available at the sender, the IAT is not, and so we need to make some assumptions to compute the IAT.

The sender sends a packet with a sequence number, and the receiver sends an acknowledgment back for that sequence number. We assume that it is possible to piggyback the IAT value in the acknowledgment, and that the link between the receiver and sender is not saturated and error-free. When sender receives an acknowledgment for a sequence number, it calculates the RTT and saves the inter-arrival time sent by the receiver. It then uses these values and same data from previous sequence numbers to estimate whether the current packet was in a congested buffer.

4.1 Learning algorithm

Given a particular kind of network and some random background traffic, we use small bursts of probe packets at periodic intervals to learn the value of the ECN at time t_i. We assume to have some limited information available from n number of earlier times t_j, where $j < i$, when previous packets belonging to the probe were received. The amount of historical data stored is bounded by memory constraints.

As mentioned before, we rely on two quantities for each historical packet at an earlier time t_j: the Round Trip Time (d_j) of a previous packet, and its inter-arrival time $t_j - t_{j-1}$. The ECN bit, $\hat{b}_i$ at time t_i, is essentially modeled as a mapping from a multiple of these two quantities computed at several past time instances, to 0 or 1 labels (congestion or no congestion). This is a 2-class classification problem, for which supervised learning algorithms, such as Support Vector Machines (SVM), can be applied. However, from a practical viewpoint, the training information required for supervised learning algorithms may not be available at the sender. For that reason, we exploit unsupervised-learning solutions. We used the K-means clustering algorithm to study its potential to learn congestion and no congestion for each transmitted packet.

Given that we have information stored for n past probe packets received, an important parameter in our prediction model is the *history length* (H). This is the number of past occurrences of received probe packets to use as input for prediction. Congestion builds up in a router over a period of time, which can be very small or very large. Therefore, we believe it is important to experiment with varying the history length in order to avoid either over-fitting or under-fitting the model. The longer the history length, the larger the amount of past packet information that is used in the prediction,

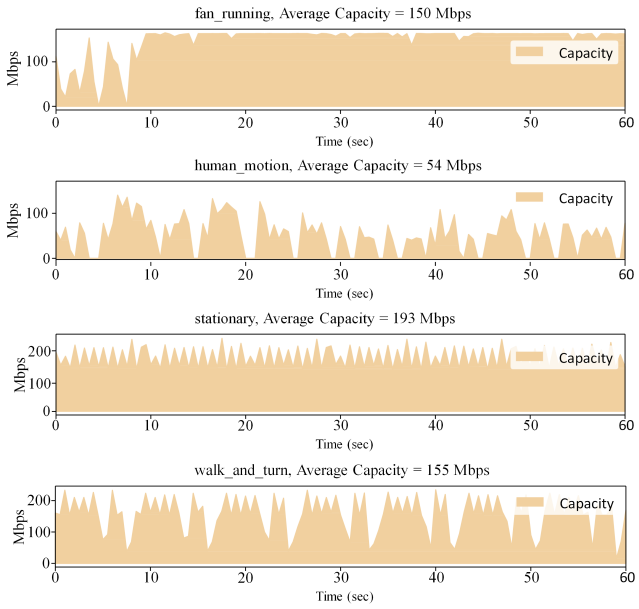


Figure 1: Capacities of various channels

- The *fan_running* trace shows the capacity when we kept the laptop and router on a desk inside a room to collect a long stable trace. The slight instability in this trace can be due to a running fan that was between the router and the laptop. The initial instability can be attributed to the human who started the trace collection script before moving away from the laptop.
- The *human_motion* trace shows the capacity of the link where we introduced human interference in between the router and the receiving laptop. As can be observed, the channel capacity is greatly affected by the attenuation due to the human body.
- The *stationary* trace shows the capacity of the trace where the laptop was placed next to the router. The y-axis shows that this is the highest capacity trace with an average capacity of 193 Mbps. This trace was collected in a different indoor environment compared to the previous ones (*fan_running* and *human_motion*).
- The *walk_and_turn* trace shows the channel capacity when a person holding the laptop moves towards the WiGig router and then turns. This trace was smaller in length so it was concatenated multiple times to produce a longer trace to emulate a long running flow.

and consequently the larger the number of parameters to estimate. Hence using a history greater than 1 may also help to ensure that the problem is not over-determined thus potentially improving learning performance.

5 EVALUATION AND RESULTS

5.1 Experimental Setup

To collect the WiGig traces, we used the NETGEAR Nighthawk X10 wireless router, which implements the 802.11ad standard [2] for 60 GHz channels. We used an Acer TravelMate P648 which comes with a WiGig enabled network card. We started a UDP sender at a Linux machine connected to the wireless router over Ethernet, and used it to send UDP packets at a very fast rate to the IP address of the laptop. A receiver at the laptop recorded the timestamp every time a packet was received. We used the log from the laptop to record the inter-arrival times between the packets. This time was used to create a trace file that is afterwards used as the channel trace in the Mahimahi [10] linkshell emulator. The trace files contains the number of slots that are available in any given millisecond to send a packet. The linkshell in Mahimahi acts as a controlled router that queues packets and sends them at the desired rate as dictated by the trace file. The traces gave us the ability to run several algorithms and compare the performance across traces. The same approach has been used by Verus [17] and Sprout [15] in the past.

We collected the WiGig traces under different scenarios. We show these traces here to demonstrate how a WiGig channel looks like under different scenarios. There were sometimes long unexpected disconnects in the collected traces. To make sure that these long disconnects were not affecting our whole analysis, we removed disconnects from traces if they were longer than 3 seconds. Figure 1 shows the capacities of the 4 traces we used. The traces are described below.

After collecting these traces, we emulated these links in the Mahimahi linkshell emulator [10], where a realistic propagation delay of 20 ms was added. Each time we emulated a link, we sent packet probe bursts of length 10 packets at periodic intervals with random Poisson traffic in the background. These packet probe bursts were sent by a sender. We varied two parameters – the interval between two consecutive probe bursts (*burst interval*), and the mean bit rate of the background traffic. We show results with two burst interval values, 5 ms and 10 ms, and the mean value of background traffic rate was between 620 and 680 Mbps. These high sending rates were chosen in order to reduce the skew between number of congestion instances and number of no-congestion instances in the training data ("class imbalance"). With background traffic rate set in the ballpark of the channel capacity, the amount of skew was as high as 1:1000, which resulted in bad performance by the ML algorithm in predicting instances of congestion correctly. Hence we kept increasing the rate until the amount of skew became acceptable. The worst skew in the current setup was approximately 1:10. We collected comprehensive data to analyze the clustering performance for each link by emulating it several times.

The dataset for the clustering model was built in the following way. From the output trace of the emulation for 60 seconds, we computed the ECN at every millisecond since the start, from the queue occupancy (recall from §4 that we mark ECN as 1 in that millisecond if the buffer occupancy was 50% or more). At every millisecond, for history length H, the RTT and IAT were taken for H immediately previous instances of packet arrivals as the input features to predict congestion at the current time. The total number of features were thus $2H$ for history length H. All such samples were collected in a dataset for each value of H and channel type. A single dataset had approximately 25000 feature-label pairs; the exact number depended on the history length H.

5.2 Learning congestion using clustering

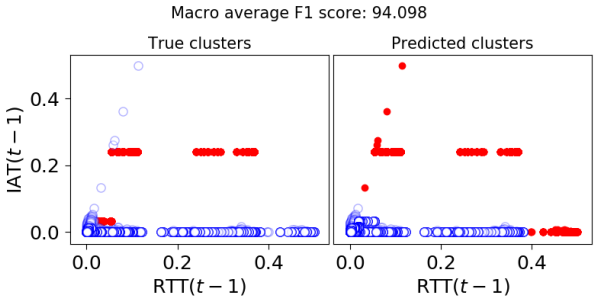
After analyzing the datasets obtained using the procedure described above, we made two observations. First, the RTT and IAT samples were on different scales. Secondly, most of the samples were confined to small ranges of values. Hence we applied the following preprocessing steps to the data before running the clustering.

(1) The feature values were transformed by a simple exponentiation, so that those having small values remain small, while the ones larger get separated from the rest. The idea of this transformation was to effect better separation of those samples indicating congestion from the rest.
(2) Outliers detected using a fixed deviation from the mean were removed. We experimented with deviation of σ, 3σ and 5σ, where σ represents the standard deviation. Outliers correspond to those samples that have large feature values, and are thus more easily separable.
(3) Finally, the resulting dataset was scaled so that RTT and IAT samples become comparable. Each feature was scaled to be in $[0, 1]$. Additionally, we assign RTT and IAT different relative weights, β and $1 - \beta$ respectively, where $\beta \in [0, 1]$.

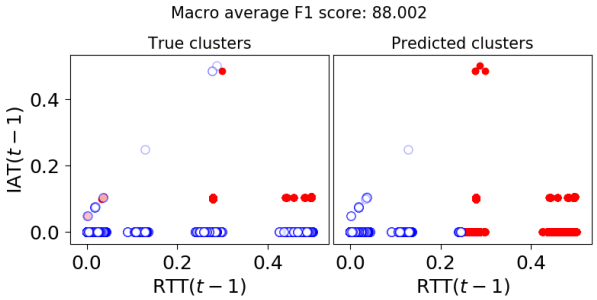
We ran the K-Means clustering with $K = 2$ separately for each burst interval, history length (from 1 to 4) and each channel type. We also experimented with the weight β in steps of 0.1 from 0 to 1. For each setting, we set aside 25% of the dataset for testing. Once a clustering model was learnt on the training set, it was applied on the testing set. One of two resulting clusters was assigned to 0 and other to 1. The assignment was done in such a way as to maximize amount of overlap with the ground truth. Following this, we are able to study clustering performance using the standard metrics used for quantifying classification performance – *precision* and *recall*. The *precision* for a class C is the fraction of samples correctly classified as class C to all the samples classified as class C, and the *recall* is the fraction of samples correctly classified as C to all the samples that are in fact class C. Compared with the *accuracy*, which is simply the fraction of number of correctly classified samples to the total number of samples, which can be misleadingly high when there is significant class imbalance as in our case, the precision and recall better capture the false positive and false negative rates in the classification. We report the *F1-score* in our results below, which is the harmonic mean of the precision and recall [13]. The F1-score is a popular and widely used metric in the field of information retrieval for measuring classification performance in many applications, as it captures both the precision and the recall [3].

Table 1 shows the best prediction result for each channel, history length and burst interval. Each reported F1-score in the table above is the average of two F1-scores, one for each of the two classes ECN=0 and ECN=1. Since our objective is to analyze the potential performance of using an unsupervised method for ECN prediction, each entry is the maximum score across all values of β. Figure 2 illustrates clustering for history $H = 1$ when feature space is two-dimensional and when $\beta = 0.5$. We note that we observe F1-scores higher than 71% even in a default setting where there is no relative weighting.

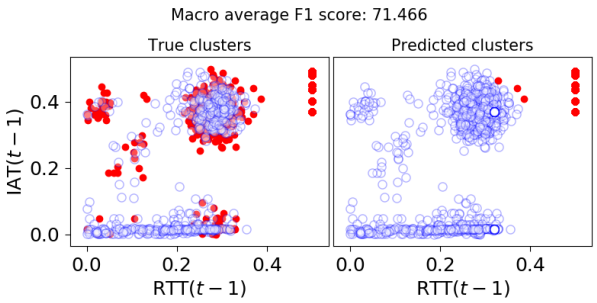
The clustering algorithm, in all cases, is able to learn the ECN with high accuracy. In addition, we can observe that, for our datasets, similar performance can be achieved irrespective of the history



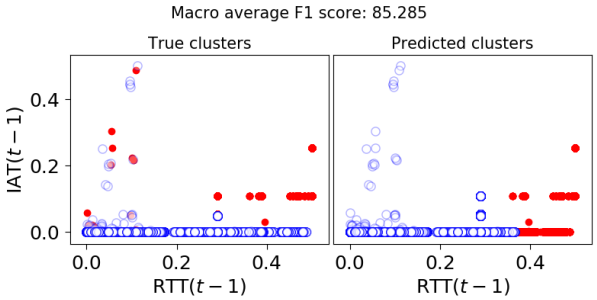
(a) *fan_running*



(b) *human_motion*



(c) *stationary*



(d) *walk_and_turn*

Figure 2: Illustration for equal weighting of RTT and IAT for all channel types for history $H = 1$ and minimal outlier removal. The red markers indicate ECN=1, and blue markers ECN=0. The illustration is to show some of the natural separation in the feature space.

Table 1: Percentage of the average F1-scores of unsupervised learning

Channels	Burst Interval 5 ms History 1	2	3	4	10 ms History 1	2	3	4
fan_running	94	94	94	94	94	94	94	94
human_motion	96	97	97	97	94	94	94	94
stationary	97	97	97	97	96	97	97	97
walk_and_turn	95	95	95	95	94	94	94	94

length. Finally, the results also indicate that the scenario with smaller inter-burst interval (i.e. 5 ms) brings about slightly better results. This last observation is due to higher congestion when the interval between traffic bursts is reduced, which in turn reduces the imbalance between the numbers of samples in each of the two classes, ECN=0 and ECN=1.

5.3 Using learned ECN

We have shown that we can learn the ECN signal for each packet with a reasonable accuracy. This learned signal can be used as an indicator of congestion building up in the network. We wanted to see if our learned signal can give us a congestion window that highly correlates with the capacity in the network at that time. To this end, we started with the DCTCP algorithm, which is designed specifically for data centers and uses the ECN. We decided to replace the true ECN signal with the learned ECN signal in the control loop of the algorithm. If the learned ECN can give us a congestion window that correlates well with the channel capacity, we can say that learned ECN can be used as a replacement for the true ECN. A high positive correlation would mean that we have a sending rate that increases and decreases proportional to the channel capacity.

For this simulation we assumed that we are in the congestion avoidance phase. Recall that congestion avoidance is the phase after slow start and TCP behavior is Additive Increase and Multiplicative Decrease (AIMD), for no-congestion and congestion respectively. In congestion avoidance phase, DCTCP does additive increase in the case of no congestion, while doing a fractional decrease based on ECN in the case of congestion. The equation for fractional decrease is: cwnd = cwnd $\times \left(1 - \frac{\alpha}{2}\right)$, where α is the weighted fraction of ECN marked packets in the last RTT. When α is close to 0 (low congestion), the window is only slightly reduced. When α is zero, the DCTCP rate only increases by the Maximum Segment Size (MSS), which is not desirable for mmWave channels but we still use it to demonstrate how learned ECN signal can be used.

We start our simulation with an arbitrary congestion window post slow start phase. After that, we calculate congestion window using the fraction of learned ECN packets in a time window (simulating RTT). We then compute the cross-correlation of the congestion window with the actual channel capacity. We observed a maximum Pearson correlation coefficient of 0.622 between the congestion window and channel capacity. This means that the congestion window shows decent correlation with the channel capacity and can be used to track channel capacity over time. DCTCP is not the perfect fit for mmWave channels because it has been designed for data centers but the correlation value indicates that the learned ECN may be a powerful signal for congestion control. We believe that we can get an even higher correlation if we replace the additive increase with a more aggressive ramp-up function. Determining the exact function that would work for mmWave links remains part of future work.

6 CONCLUSION

Our results are a first attempt at solving the problem of learning congestion state of network, and we believe that these results are very encouraging. The fact that we are able to use unsupervised learning and obtain such high accuracy shows the immense potential of this approach to replace or augment traditional ECN and other congestion notification methods. In our experiments, we also tried other ML algorithms such as logistic regression and SVM, but found that a simple unsupervised learning approach like K-means performs the best. In future work we intend to explore the use of reinforcement learning, as well as looking at the exact control loop for congestion control and online ECN learning.

ACKNOWLEDGMENTS

This work was supported by Defense Advanced Research Projects Agency (DARPA) contract HR001117C0048. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. Luis Diez & Ramón Agüero were partially supported by the Spanish Government (Ministerio de Economía y Competitividad, Fondo Europeo de Desarrollo Regional, FEDER) by means of the projects ADVICE: Dynamic provisioning of connectivity in high density 5G wireless scenarios (TEC2015-71329-C2-1-R) and FIERCE: Future Internet Enabled Resilient Cities (RTI2018-093475-A-100).

REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. ACM, New York, NY, USA, 63–74.

[2] IEEE Standards Association et al. 2014. *IEEE Std 802.11 ad-2012: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.* Technical Report. ISO/IEC/IEEE 8802–11: 2012/Amd. 3: 2014 (E).

[3] Steven M. Beitzel. 2006. On Understanding and Classifying Web Queries. (2006).

[4] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. 2005. Design and Implementation of a Routing Control Platform. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*. USENIX Association, Berkeley, CA, USA, 15–28. http://dl.acm.org/citation.cfm?id=1251203.1251205

[5] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. {PCC} Vivace: Online-Learning Congestion Control. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 343–356.

[6] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking* 1, 4 (1993), 397–413.

[7] Pierre Geurts, Ibtissam El Khayat, and Guy Leduc. 2004. A Machine Learning Approach to Improve Congestion Control over Wireless Computer Networks. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM '04)*. IEEE Computer Society, Washington, DC, USA, 383–386. http://dl.acm.org/citation.cfm?id=1032649.1033486

[8] Zhenxian Hu, Yi-Chao Chen, Lili Qiu, Guangtao Xue, Hongzi Zhu, Nicholas Zhang, Cheng He, Lujia Pan, and Caifeng He. 2015. An In-depth Analysis of 3G Traffic and Performance. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges (AllThingsCellular '15)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/2785971.2785981

[9] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM computer communication review* 32, 4 (2002), 89–102.

[10] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP.. In *USENIX Annual Technical Conference*. 417–429.

[11] Yong Niu, Yong Li, Depeng Jin, Li Su, and Athanasios V Vasilakos. 2015. A survey of millimeter wave communications (mmWave) for 5G: opportunities and challenges. *Wireless Networks* 21, 8 (2015), 2657–2676.

[12] K. Ramakrishnan, S. Floyd, and D. Black. 2001. *The Addition of Explicit congestion Notification (ECN) to IP.* RFC 3168. Network Working Group. 1–63 pages. https://tools.ietf.org/html/rfc3168

[13] C. J. Van Rijsbergen. 1979. *Information Retrieval* (2nd ed.). Butterworth-Heinemann, Newton, MA, USA.

[14] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan. 2014. An Experimental Study of the Learnability of Congestion Control. In *Proceedings of the ACM SIGCOMM 2014 Conference*. Chicago, IL, USA.

[15] Keith Winstein, Anirudh Sivaraman, Hari Balakrishnan, et al. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks.. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL.

[16] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 731–743.

[17] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg. 2015. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the ACM SIGCOMM 2015 Conference*. London, UK, 509–522.