Learning-based Congestion Control for Internet Video Communication over Wireless Networks

Tongyu Dai[1], Xinggong Zhang[1,2] and Zongming Guo[1,2,§]
[1]Institute of Computer Science & Technology, Peking University, Beijing, P.R. China, 100871
[2]Cooperative Medianet Innovation Center, Shanghai, China
{dai_tongyu, zhangxg, guozongming}@pku.edu.cn

***Abstract*—With the deployment of real-time video applications and wireless networks, the real-time congestion control becomes a hot topic. Most existing congestion control algorithms are not designed for low-latency real-time flows, or perform poorly in the face of highly variable channel capacities. In this paper, we proposed a novel Learning-based Congestion Control (LCC) for real-time video communication over wireless networks. The key idea of LCC is employing Kernel Density Estimation for one-way delay and sending rate to capture the underlying information about channel state. Then LCC bases on the estimated probability density and Bayesian theorem to quickly adapt sending rate to the changing channel. We implemented LCC in WebRTC framework and extensive experiments were carried out. Compared with the native WebRTC congestion control (GCC), experimental results show that LCC achieves higher channel utilization, even more than 4.2$\times$ throughput in lossy links. LCC is also much better at adapting to the variable channel than GCC. Besides, LCC performs well in delay constraint and intra-protocol fairness.**

I. Introduction

With the advance of mobile network and video technology, interactive video applications are increasingly emerging and wireless networks have become a popular mode of Internet access. As a consequence, real-time congestion control becomes a hot topic. Two IETF working groups, RTCWeb and RMCAT, are standardizing a set of protocols for real-time video communication. There are also increasing researchers devoted into real-time congestion control research.

Although there has been already many prior works in the field of congestion control, it is still a challenge to design an efficient algorithm for real-time video streaming over wireless networks. The effects of wireless link-level mechanisms on end-to-end transport protocols were well studied in [1] and the challenges can be summarized in two parts: rapid variation of the link capacity and bursty [2].

The conventional algorithms like TCP Cubic [3] and TCP Vegas [4] are not usually employed by real-time flows. They use constant parameters to adjust the rate, introducing rate oscillation and intolerable queuing delay [5]–[7]. Similarly, the algorithms designed for real-time video streaming, such as TFRC, GCC and NADA [8]–[10], also have some serious flaws. They all back off when packet loss occurs, but none of them can distinguish stochastic loss which is a part of the wireless environment from the packet loss caused by congestion, resulting in bandwidth under-utilization. Besides, there are several recent algorithms designed for various environments, such as PCC [11], Sprout [12], Verus [2] and Remy [13]. Especially, Verus and Remy novelly try to learn the control strategy, separately based on online observation and offline prior knowledge. But Remy can not perform well in the network beyond the training set.

Motivated by these issues, we aim to design a specialized congestion control for real-time video streaming over wireless networks, to achieve following goals: 1) low queuing latency; 2) high channel utilization; 3) adaptability to changing channel; 4) a fair share of bandwidth between concurrent flows.

Addressing the above goals, we designed a novel end-to-end Learning-based Congestion Control (LCC), specialized for real-time video communication over wireless networks. The key of LCC is employing weighted Kernel Density Estimation (KDE) to continuously learn the relationship between sending rate and one-way delay (OWD), which implies the information about wireless channel condition. Then, based on the difference between OWD and our desired delay target, LCC employs Bayesian theorem to adaptively adjust sending rate. This approach avoids the effects of fixed adjustment parameters, achieving higher channel utilization and better adaptability to the changing channel.

We implemented our proposed LCC algorithm in the WebRTC framework, and experimentally evaluated LCC in the typical scenario described in the IETF RMCAT [14]. By comparing with the native WebRTC congestion control (GCC), the experimental results show that LCC 1) obtains higher channel utilization, even more than $4.2\times$ throughput when stochastic loss rate is high, at the cost of only a dozen milliseconds delay; 2) achieves smoother bitrate and faster convergence; 3) gets a fair share of bandwidth between concurrent LCC flows.

The main contributions of this paper are three-fold:

- A probability density model is used to objectively reflect the channel status, instead of attempting to predict the unpredicted wireless channel dynamics.
- Based on Bayesian theorem, the sending rate is quickly adjusted to achieve high channel utilization and the adaptation to changing link.
- A delay target is introduced to successfully constrain the queuing delay in bottleneck links.

[§]Corresponding author of this paper.

This work was supported by National Natural Science Foundation of China under contract No. 61471009 and Beijing Culture Development Funding under Grant No.2016-288.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

This paper is organized as follows: Section II describes the components of LCC. Section III shows the experimental environment, results and corresponding analysis. Section IV concludes the paper.

II. LCC Algorithm

LCC is an end-to-end congestion control protocol designed for Internet real-time video communication over wireless networks. Due to the unpredictability of wireless networks, LCC uses OWDs to reflect the state of bottleneck link. The key idea of LCC is to continuously learn a probability density model which captures the relationship between OWDs and sending rate. Then LCC employs this relationship to increase or decrease sending rate, based on the desired delay target and Bayesian theorem.

Considering that the channel changes over short time scales in wireless networks, LCC adjusts sending rate in a small ε ms cycle to quickly adapt to the changing link. At each epoch, LCC decides the next sending rate S_{i+1} as follows:

$$S_{i+1} = f(D_{ave,i}, P(S,D)) \tag{1}$$

where f is the rate decision function based on Bayesian theorem, with $D_{ave,i}$ being the average OWD at i-th epoch and $P(S,D)$ representing the joint probability density of sending rate S and OWD D.

Therefore, to build the rate decision function f, LCC first uses Delay-Rate Tracker to estimate and record historical sending rate and OWD, then employs Probability Density Fitter to obtain the joint probability density of sending rate and OWD. Finally, it constructs f with Bayesian Controller.

A. Delay-Rate Tracker

The Delay-Rate Tracker aims to estimate and record the sending rate and average OWD at each epoch. OWD is calculated by the timestamp difference between receiver and sender, which helps to avoid the influence caused by reverse cross-traffic.

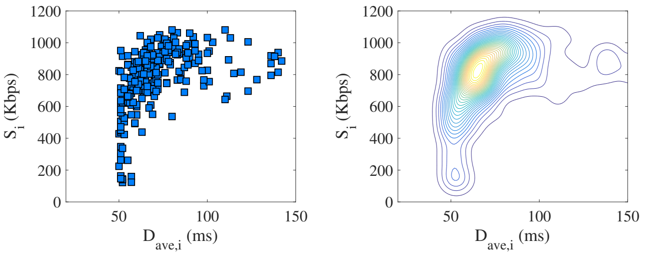
Within the i-th epoch, Delay-Rate Tracker keeps track of all received OWDs, then figures out the temporary average value D_i. In order to avoid abrupt changes, the average OWD $D_{ave,i}$ is weighted by an Exponential Weighted Moving Average (EWMA) as follows:

$$\begin{gathered} D_{ave,i} = (1-\alpha)D_{ave,i-1} + \alpha D_i \\ where \quad 0 < \alpha < 1 \end{gathered} \tag{2}$$

Besides, the sending rate S_i at i-th epoch is also recorded by Delay-Rate Tracker. Finally, all such pairs of $D_{ave,i}$ and S_i are passed to Probability Density Fitter.

B. Probability Density Fitter

In a bottleneck link, the sending rate S_i can be viewed as an "action" and $D_{ave,i}$ is the corresponding "result". The pair of S_i and $D_{ave,i}$ implicitly reflects the channel state. Employing many such pairs, the Probability Density Fitter aims to describe the latest channel status in probabilistic terms.



(a) Collected data. (b) Contour lines of P(S, D).

Fig. 1: Probability Density Estimation for $D_{ave,i}$ and S_i.

The Probability Density Fitter sets a sliding window to store the latest N pairs of S and D, to establish the joint probability density. In addition, the recent pairs are given more weight than the old. For $1 \leqslant i \leqslant N$, the weight of the i-th pair is computed as follows:

$$w_i = e^{i/N-1} \tag{3}$$

The rationality of the weight is obvious that the more recent the data is, the better it is for demonstrating the channel state.

Based on above sampled datas, bivariate weighted Kernel Density Estimation (KDE) is used to fit the joint probability density of S and D. The probability density function $P(S,D)$ is estimated as follows:

$$\begin{gathered} P(S,D) = \frac{1}{W}\sum_{i=1}^{N} w_i K(S-S_i, D-D_{ave,i}) \\ where \quad W = \sum_{i=1}^{N} w_i \end{gathered} \tag{4}$$

In equation 4, K represents the kernel function and the normal kernel is adopted in this paper.

The joint probability density captures the relationship between OWD and sending rate, describing the latest channel state in probabilistic terms. To show its function intuitively, we collected data based on the test-bed built in Section III-B, where N is set to 300. The resulting probability density estimation displayed in Figure 1 is logical, where larger sending rate will cause higher delay. The region with the largest probability is exactly the area LCC converges to.

Considering the computational complexity, the probability density update interval is set to 1 second. Finally, $P(S,D)$ is passed to Bayesian Controller to decide the next sending rate.

C. Bayesian Controller

Bayesian Controller employs OWD information and joint probability density $P(S,D)$ to decide the next sending rate. In order to adapt quickly to channel changes, the update interval of sending rate is set to a small ε milliseconds.

In order to effectively constrain the packet delay, we set a low delay target T_A as the goal of following OWDs. Through grid search technique, the minimum OWD plus 25ms is proved to be a suitable value for T_A. Delay control parameter T_c is used as the reference delay to adjust sending rate. Within i-th epoch, $T_{c,i}$ is expressed as follows:

$$T_{c,i} = T_{c,i-1} + \delta(T_A - D_{ave,i}) \tag{5}$$

where δ is a positive increment/decrement parameter.

Then, Bayesian Controller aims to find the suitable sending rate corresponding to the delay control parameter $T_{c,i}$. It is based on Bayesian theorem and the law of total probability:

$$\begin{aligned} P(S|D = T_{c,i}) &= \frac{P(S, D = T_{c,i})}{P(D = T_{c,i})} \\ P(D = T_{c,i}) &= \int P(S, D = T_{c,i}) d(S) \end{aligned} \tag{6}$$

The expectation of the sending rate which caused OWD equal to $T_{c,i}$ is chosen as the next sending rate:

$$S_{i+1} = \int S \cdot P(S|D = T_{c,i}) d(S) \tag{7}$$

In addition, LCC deals with the exorbitant delays (i.e. larger than $R\times$ T_A) by a multiple parameter M:

$$S_{i+1} = M \cdot S_i \qquad if\ D_{ave,i} > R \cdot T_A \tag{8}$$

The rationality of Bayesian Controller is obvious: if the average OWD estimate $D_{ave,i}$ is higher than delay target T_A, $T_{c,i}$ should be decreased to choose a lower sending rate, and vice versa.

III. Algorithm Performance

We implemented LCC in WebRTC framework[*] and conducted extensive experiments to evaluate the performance of LCC. In [15], it is revealed that NADA and GCC have similar performance. Thus we only compared LCC with the native WebRTC congestion control GCC.

A. Experiment Setup

To verify our proposed algorithm experimentally, we set up experimental test-bed over a real network consisting of two Windows machines connected through Ethernet. Machine-1 is used as a sender while Machine-2 is a receiver, and they both have a private IP address. The WebRTC clients we used are compiled from native code in *branch M60*. To generate video flows, Machine-1 runs the *peerconnection_client.exe* as a client and *peerconnection_server.exe* as a server, while Machine-2 runs the *peerconnection_client.exe* as a client. To support the experiments reproducibility, we used the same video sequence for every emulation, employing the virtual webcam *VCam*. Besides, *Network Emulator* is used to set network parameters, such as available bandwidth, link loss rate and so on. The one-way propagation delay was set to 50 milliseconds on the forward path. To emulate the deep queue in wireless networks, we did not limit the queue size. Finally, *Wireshark* is used to capture packets.

[*]https://webrtc.org/

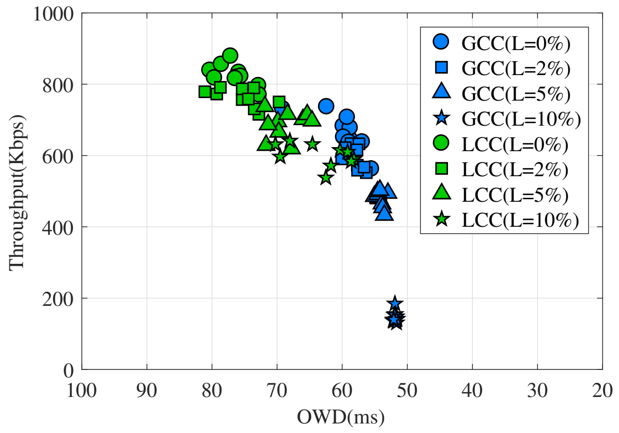


Fig. 2: Average throughput and OWD for each flow of LCC and GCC under different stochastic loss rate, over the $1000Kbps$ link.

We consider the following metrics to evaluate the performance of algorithms: 1) Channel Utilization $U = \bar{x}/b$, where b is the known available bottleneck bandwidth and $\bar{x}$ is the average throughput; 2) Queuing delay T_q, estimated by the average of the difference between OWDs and propagation delay.

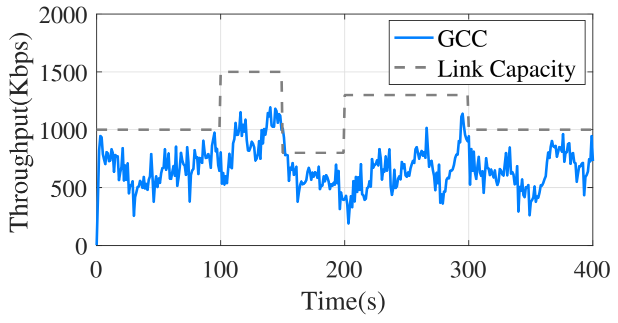
B. Single flow in scenarios with different loss rate

This experiment aims to investigate the performance of single LCC and GCC over a lossy link. To emulate real-world wireless environment, the bottleneck link is configured with different packet loss rate $l_i \in [0, 2\%, 5\%, 10\%]$, with available bandwidth b equal to 1Mbps. Each experiment is repeated ten times and the duration of each test is 100 seconds.

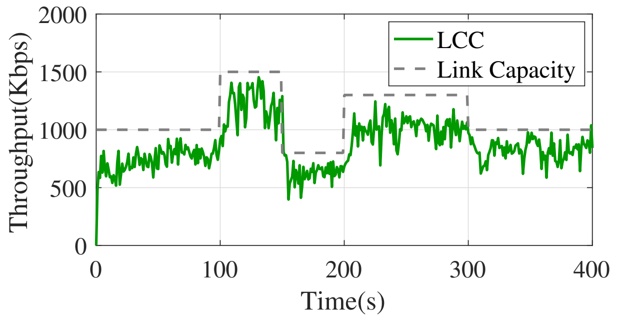
TABLE I: Average channel utilization U and queuing delay T_q over the link with different stochastic loss rate.

Loss Rate		0%	2%	5%	10%
$U(\%)$	GCC	69.2%	59.5%	47.8%	14.6%
	LCC	82.4%	77.3%	70.6%	61.6%
T_q(ms)	GCC	9.95	8.11	4.14	1.87
	LCC	25.5	23.3	17.6	13.3

Figure 2 shows the average throughput and OWD for each of the flows across all tests. It demonstrates that LCC always performs better than GCC under different stochastic loss rate. Although the queuing delay of GCC is exactly low, it wastes too much available bandwidth and always keeps the channel underutilized. Table I displays the statistical average value of channel utilization U and queuing delay T_q for every scenario. It clearly shows that LCC achieves more than 18% throughput than GCC at the cost of only a dozen milliseconds queuing delay. Even when the stochastic loss is up to 10%, the throughput enhancement is more than 3.2 times.



(a) A single GCC flow.



(b) A single LCC flow.

Fig. 3: Throughput dynamics in the case of a single GCC or LCC flow with variable link capacity.

C. Single flow with variable link capacity

This scenario investigates the dynamics of the bitrate controlled by GCC or LCC to a step-like variation of the link capacity. To the purpose, we set the link capacity to 1Mbps for the fist 100s, then we increase it to 1.5Mbps for 50s and decrease it to 800Kbps for another 50s. The following link capacity is set to 1.3Mbps for 100s and back to 1Mbps for the last 100s. Besides, the packet loss rate is set to 0%.

Figure 3 displays the dynamic throughput of GCC and LCC. It intuitively shows that, in the face of varying link capacity, LCC quickly adapts its sending rate to channel dynamics, while the response of GCC is so conservative that it wastes too much available bandwidth. The bad performance of GCC is caused by the long adjustment interval and sensitive delay threshold. On the contrary, LCC acts over the small epoch to adapt quickly to network changes. In wireless networks, the channel changes more violently, where GCC will perform much worse. Besides, the throughput of LCC is also more stable than GCC.

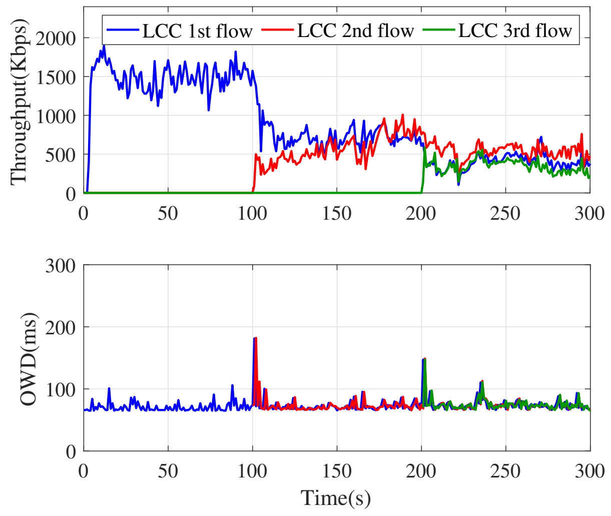


Fig. 4: Throughput and OWD dynamics in the case of three LCC flows sharing a bottleneck link with $2Mbps$.

D. Intra-protocol fairness

The aim of this scenario is to investigate the intra-protocol fairness of our proposed LCC. To the purpose, we have considered three concurrent LCC flows over a 2Mbps link. Each flow is started 100 seconds after the previous one. The emulation lasts 300 seconds in total. Besides, the stochastic loss rate is set to 0% and the one-way propagation delay is still 50 milliseconds.

Figure 4 shows the overlapping dynamics of LCC throughput and OWD. We can notice that LCC is not affected by other competitors and nicely shares the bandwidth among the flows. As far as delay is concerned, OWDs are always kept around 75ms, i.e. 25ms queuing delay. Only when a new comer enters, the queuing delay increment is somewhat large, which is a normal phenomenon.

IV. Conclusion

In this paper, we have designed a novel end-to-end congestion control algorithm named LCC, specialized for the real-time video communication over wireless networks. Instead of attempting to predict the unpredicted wireless channel dynamics, LCC bases on a probability density model, which captures the relationship between one-way delay and sending rate, to quickly adapt sending rate to the changing channel. We implement LCC in WebRTC framework and evaluate it under several experimental scenarios, by comparing with the native WebRTC congestion control (GCC). The results show that LCC has great improvements: 1) Higher channel utilization than GCC is achieved, even up to $4.2\times$ throughput when packet loss rate is high; 2) The delay is constrained well, around 20ms; 3) The great ability to adapt to the changing channel is obtained, which is much better than GCC; 4) A fair share of bandwidth is achieved in the case of several concurrent LCC flows over a link.

References

[1] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 85–96, 2004.

[2] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, “Adaptive congestion control for unpredictable cellular networks,” *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 509–522, 2015.

[3] S. Ha, I. Rhee, and L. Xu, “Cubic: a new tcp-friendly high-speed tcp variant,” *ACM Sigops Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[4] L. S. Brakmo and L. L. Peterson, “TCP Vegas: end to end congestion avoidance on a global internet,” *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 2002.

[5] J. Gettys and K. Nichols, “Bufferbloat: dark buffers in the internet,” *Communications of The ACM*, vol. 55, no. 1, pp. 57–65, 2012.

[6] K. Winstein and H. Balakrishnan, “End-to-end transmission control by modeling uncertainty about the network state,” in *ACM Workshop on Hot Topics in Networks*, 2011, pp. 1–6.

[7] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, “An in-depth study of LTE: effect of network protocol and application behavior on performance,” in *ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013, pp. 363–374.

[8] M. Handley, S. Floyd, J. Padhye, and J. Widmer, “TCP Friendly Rate Control (TFRC): Protocol Specification,” *RFC 5348*, 2008.

[9] G. Carlucci, S. Holmer, S. Holmer, and S. Mascolo, “Analysis and design of the google congestion control for web real-time communication (WebRTC),” in *International Conference on Multimedia Systems*, 2016, p. 13.

[10] S. D’Aronco, P. Jones, C. Ganzhorn, R. Pan, M. Ramalho, S. D. L. Cruz, and X. Zhu, “NADA: A unified congestion control scheme for real-time media,” in *Packet Video Workshop*, 2015, pp. 1–8.

[11] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, “PCC: Re-architecting congestion control for consistent high performance.” in *NSDI*, 2015, pp. 395–408.

[12] K. Winstein, A. Sivaraman, H. Balakrishnan *et al.*, “Stochastic forecasts achieve high throughput and low delay over cellular networks.” in *NSDI*, 2013, pp. 459–471.

[13] K. Winstein and H. Balakrishnan, “Tcp ex machina: Computer-generated congestion control,” in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 123–134.

[14] Z. Sarker, V. Singh, X. Zhu, and M. Ramalho, “Test cases for evaluating RMCAT proposals,” *draft-ietf-rmcat-eval-test-05 (work in progress)*, 2017.

[15] G. Carlucci, L. D. Cicco, C. Ilharco, and S. Mascolo, “Congestion control for real-time communications: A comparison between NADA and GCC,” in *Control and Automation*, 2016, pp. 575–580.