Congestion Control for RTP Media: A Comparison on Simulated Environment

Songyang Zhang, Weimin Lei(✉), Wei Zhang, and Yunchong Guan

School of Computer Science and Engineering, Northeastern University, Shenyang, China
leiweimin@ise.neu.edu.cn

Abstract. The audio and video applications based on Real Time Protocol (RTP) have been exploded in recent years. To develop low latency congestion control algorithms for real time traffic to provide better quality of experience and to avoid network congestion has gained much attention. RTP Media Congestion Avoidance Techniques (RMCAT) working group was initiated for proposal draft. Currently, there are three algorithms under this group, Network Assisted Dynamic Adaptation (NADA), Google Congestion Control (GCC) and Self-Clocked Rate Adaptation for Multimedia (SCReAM). This paper integrates the three algorithms into simulated platform and their performances are compared and analyzed. Results show GCC has well fairness property and can maintain quite reasonable packet sending rate in loss link but converges a bit slowly in dynamic link, NADA stabilizes its rate quickly but suffers from "late-comer" effect, SCReAM has the lowest queue occupation but also lower link capacity utilization.

Keywords: Congestion control · RTP media congestion control · Real time traffic · ns3 simulation

1 Introduction

Pioneered by Jocobson's work [1], which later developed into TCP Reno algorithm, network congestion control has been an unfading topic in computer networks research. The control law proposed by Jocobson is to regulate TCP sending rate according to additive increase and multiplicative decrease (AIMD) rule, which developed as TCP Reno. It takes packet loss as network congestion signal. On every RTT, the sender could send one more packet into network to probe more available bandwidth and multiplicatively reduces congestion window size by half when packet loss happens to alleviate link congestion.

This work was supported by the National Key Research and Development Program of China (2018YFB1702000), the Liaoning Provincial Natural Science Foundation of China (No. 20180551007), and the National Natural Science Foundation of China (No. 61671141).

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2019
Published by Springer Nature Switzerland AG 2019. All Rights Reserved
H. Song et al. (Eds.): SIMUtools 2019, LNICST 295, pp. 43–52, 2019.
https://doi.org/10.1007/978-3-030-32216-8_4

From then on, most of the research works such as Bic [2], Cubic [3], were proposed to improve TCP performance and adapted the basic AIMD control law to different network environment.

The congestion control algorithms in TCP are mainly compliant with bulk data transfer. The additive increase of TCP during its congestion avoid phase cause packet sending rate showing saw-tooth feature. If such mechanism is applied directly to real time video applications, the AIMD rate control would cause the instability of video encoder. Further, the packet lost retransmission and in order delivery in TCP would introduce further latency, which makes it unfit for time stringent traffic transmission. The real time video traffic is quite sensitive to connection latency but can suffer some packets loss to some extent. So RTP-based media is usually streamed over UDP.

In an early stage, the implementation of congestion control on UDP for video streaming is quite scarce, due to the consideration that an insufferable QoE (Quality of Experience) of connection would make the users give up video call, which can be seen as a mechanism of congestion avoidance. The network condition has changed in better direction and is used in a different manner as it was ten years ago.

If large scale video stream flows do not implement any congestion control mechanism, the bandwidth competition would lead Internet into congestion. The extra packets would be buffered in the intermediate router when the link is in congestion. Once the queue length has increased above the link queue threshold, the router would follow active queue mechanism by dropping packets and the link transmission delay will increase. Even though there were some works [4,5] making an effort to exploit congestion control for UDP streaming media before, none of these algorithms have been applied in practice.

To develop new congestion control algorithm for real time traffic has gained renewed attention in recent years, especially since the open source of Web Real-Time Communication (WebRTC) which enables real time communication between browsers. As pointed by [6], all the flows across internet should implement congestion control scheme for internet congestion avoidance and promote fair bandwidth occupation. The IETF has initiated The RTP Media Congestion Avoidance Techniques (RMCAT) Working Group to develop congestion standards for interactive real-time media. And there are mainly three congestion control drafts under this working group, namely, GCC [7], NADA [8], SCReAM [9].

In this paper, we work our way to get the three RMCAT algorithms running in ns-3[1] and make a full comparison in terms of fairness, aggressiveness, bandwidth utilization and link queue occupancy. The simulation code of NADA[2] on ns3 was already released by its author. So the work is mainly focused on the implementation of GCC, SCReAM. The simulation code of this work can be downloaded at[3].

[1] https://www.nsnam.org/.
[2] https://github.com/cisco/ns3-rmcat.
[3] https://github.com/SoonyangZhang/rmcat-ns3.

The main contribution of this work is the algorithms code transplantation on simulated platform and the final results can be taken for reference for interested readers.

The rest of this paper is organized as follows. Section 2 describes the principle of these algorithms involved in simulation in detail. Section 3 is the simulation results and analysis. The conclusion is in Sect. 4.

2 Algorithm Description

This part briefly describes the algorithms involved in our experiments. The GCC algorithm exploits one-way delay gradient as control signal. The old version of GCC has two components: a delay based congestion controller, running at the receiver side, computes a rate A_r based on Kalman filter according to frame delay signal which is fed back through RTCP Receiver Estimated Maximum Bitrate (REMB) report; a loss based controller running at sender side, computes a target bitrate A_s which shall not exceed A_r. Kalman filter is adopted at the receiver side to compute the link queue delay gradient. In newer release version of WebRTC, the congestion control logics have all been moved to the sender side. A trend line filter has been introduced for congestion inference. We refer here the old version WebRTC congestion control based on Kalman filter as REMB-GCC and the newer version based on trend-line filter is referred as TFB-GCC (transport feedback GCC). The algorithm designers have published several papers on REMB-GCC, please refer to [10] for more information. We analyze the TFB-GCC in detail considering there is no public available paper on its working principle. In GCC, the receiver will feedback packets arrival time to the sender through the RTCP extensions for transport-wide congestion control [11]. The feedback message will be sent at an adaption interval according to bandwidth. When the feedback message arrives, the sender extracts out the arriving time of a sent packet, and divides them into groups by length of five milliseconds.

The packets group is similar to the frame notation in [10] for the purpose of channel overuse detection. The time_stamp is the time sending out the first packet and complete_time is the time of last packet arriving to the destination of the same group. The *j-th* group one-way delay gradient is computed as follows:

$$\begin{aligned} delta\_ms_j = &(G_j.complete\_time - G_{j-1}.complete\_time) \\ &- (G_j.timestamp - G_{j-1}.timestamp). \end{aligned} \tag{1}$$

Then compute the accumulated delay:

$$acc\_delay_i = \sum_{j=1}^{i} delta\_ms_j. \tag{2}$$

And then smooth the delay signal with a coefficient alpha by default 0.9.

$$\begin{aligned} smoothed\_delay_i = &smoothing\_coef * smoothed\_delay_{i-1} \\ &+ (1 - smoothing\_coef) * acc\_delay_i \end{aligned} \tag{3}$$

A linear regression was carried out in trend-line filter with input values of (x, y).

$$(x, y) \Rightarrow (G_i.complete\_time - G_1.complete\_time, \\ smoothed\_delay_i). \tag{4}$$

$$trendline\_slope = \frac{\sum(x_i - x_{avg})(y_i - y_{avg})}{\sum(x_i - x_{avg})^2} \tag{5}$$

The trend line slope is a reflection of link queue status. When the link queue length increases, the inter-arriving space among packets tends to increase also. The overuse detector compares the value of trend line slope with a dynamic threshold to decide if the channel is in the state of underuse or overuse. The dynamic threshold is explained by the designer [12] to tune the sensitivity of the algorithm. A small threshold will make the detector quick detect the channel state changes but with the drawback of overreacting in case of noise. A large threshold would make the algorithm robust to noise but sluggish to channel state change. And a constant threshold would make the GCC flows starvation in competing with loss based TCP flows as reported by [13]. After the overuse detector computes out the channel state, the AIMD controller adjusts the bitrate according to the equation:

$$A(t_i) = \begin{cases} A(t_{i-1}) + \overline{A} & Increase \\ \beta R(t_{i-1}) & Decrease, \\ A(t_i) & Hold. \end{cases} \tag{6}$$

where $\beta = 0.85$, and $R(t_{i-1})$ is the average receiving rate estimated at the sender side based on feedback message. The value of $\overline{A}$ is depended on the rate control region. After the rate is decreased, the controller would set the rate control region in state of near-max. After the channel is detected underuse and the control region is in near-max state, the AIMD controller would additively increase rate, otherwise, the rate is multiplicatively increased.

There is a detailed description and comparison between GCC and NADA on the WebRTC codebase platform in [14]. Their experiment was conducted on real network testbed.

The NADA has experienced several updates since its original release [8]. Basically, the NADA algorithm control its packet sending rate according to an aggregated congestion signal as shown in Eq. (7). p_{loss} and p_{mark} are the penalty prices when a sent packet is dropped or marked by the intermediate router to indicate the link in congestion status. The mark signal has the same purpose as the explicit congestion notification in TCP to enable end to end congestion notification without dropping packets.

$$x_n(t_i) = \tilde{d}(t_i) + p_{loss} * D_{loss} + p_{mark} * D_{mark} \tag{7}$$

Here, the term $\tilde{d}(t_i)$ is computed as Eq. (8) and QTH is 50 ms. d_q is computed as the following. The receiver would send feedback message every 100 ms. The feedback message contains the timestamp R_{ts} that a sent packet arrives the

destination. The sender would record the history packets sent within 500 ms and could compute the one way delay value $d(i) = R_{ts} - S_{ts}$. Then one way delay variance is computed $owdv(i) = d(i) - d_{min}$, which is an indication of the link congestion status. d_{min} is the minimal one way delay during the session. And the minimal $owdv$ during the last 500 ms is assigned to d_q.

$$\tilde{d}(t_i) = \begin{cases} d_q & d_q < QTH \\ QTHe^{-\lambda \frac{d_q - QTH}{QTH}} & \text{otherwise} \end{cases} \tag{8}$$

The rate control law of NADA is a piece function shown as Eq. (9). According to the link status, it would follow different control function. When the session is just established, $rmode = 0$, the sender is in accelerated ramp up state, and the sending rate is the product of the computed packet receive rate and a gain $1+\gamma$, where $\gamma = min(0.5, \frac{QTH}{rtt+\delta})$. The rate gain is exploited for available bandwidth probe purpose. The accelerated ramp up state is quite similar to TCP slow start phase. Once a packet loss event is detected during this observation period or the one way delay variance exceeds 10 ms, the sender would enter into the gradual rate update state.

$$R_n(t_k) = \begin{cases} (1+\gamma)R_n(t_{k-1}) & \text{rmode} = 0 \\ R_n(t_{k-1})(1 - K_1 x_o(t_k) - K_2 x_d(t_k)) & \text{rmode} = 1 \end{cases} \tag{9}$$

And $K1$, $K2$ are two constants, and x_o, x_d are the offset value from a reference x_{ref} and difference value, where x_{ref} is 10 ms.

$$x_o(t_k) = x_n(t_k) - x_{ref}\frac{R_{max}}{R_n(t_k)} \tag{10}$$

$$x_d(t_k) = x_n(t_k) - x_n(t_{k-1}) \tag{11}$$

The rate control function of NADA in the gradual rate update state is similar to a PID (proportional–integral–derivative) controller. If the aggregate congestion signal has a decrease tendency ($x_d < 0$), the sender would increase its packet injecting rate according to the control function. When the current rate decreases to a small value, the component x_o would control the sender to inject burst packets into network to make a quickly convergence to the available rate. When the sending rate approaches the link available rate, the action of rate increase would link queue length increase and the extra sent packet would at a high risk in lost, which would make the control term x_o, x_d positive value, and NADA sender would actively decrease its rate to avoid congestion and self-inflicted queue delay. Such feature of NADA makes highly network bandwidth utilization, which is verified by simulation results.

SCReAM basically controls the upper limit packets in flight by sliding congestion window. The receiver will feedback the timestamp of received packet with the highest sequence number and an acknowledgement vector to indicate the reception or loss of previous packets. Its congestion control method based on queue delay signal was inspired by LEDBAT, which has claimed for the low

network queue delay purpose by inferring congestion earlier. When the one-way queue delay under the target queue delay, the algorithm will increase the congestion window, otherwise decrease the window.

3 Simulation Comparison

The version of simulation platform is ns-3.26. A point to point link as suggested by [15] was created with link bandwidth 2 Mbps, one way delay 100 ms and buffer length (300 ms * 2 Mbps).

3.1 Protocol Responsiveness

Considering the popularity of mobile devices, the RTP-based media over mobile phone is quite common. The cellular access network link can present drastic change in channel bandwidth in a short time span due to noise interference and fading. The rate control algorithm for conversational video over wireless links should react quickly to network change and operates in a wide range of bandwidth. When the link bandwidth decreases, the video generator keeping the rate before would make the link queue build up and the end latency increase. When the link bandwidth increases, the sender could not make fully use of network resource if the rate do not change. The increased latency, the packet dropping event and the low video encoding rate is harmful to QoE for users.

In experiment, the link bandwidth is changed every 20 s from 500 kbps to 2 Mbps. The link is exclusively occupied by a single GCC, NADA, SCReAM. During the simulation process, the rate adjustment of the congestion control algorithm is logged and the one way delay of received packet was recorded. The one way delay is an indication of link queue occupation. When the link is in congestion status and the sent packet would be queued in router buffer, which results in high one way delay. The results in Fig. 1 have clearly shown the reaction difference of these protocols when link capacity changes. The AIMD controller in GCC for rate adjustment is the reason of its rate saw-tooth feature. When the link capacity decreases, GCC makes a quick rate adjustment to prevent link

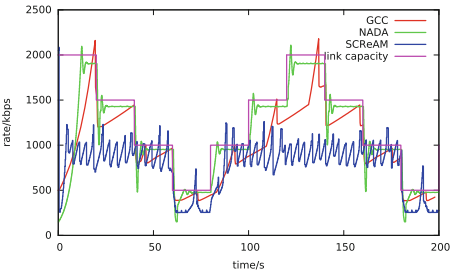

Fig. 1. The responsiveness of three algorithms

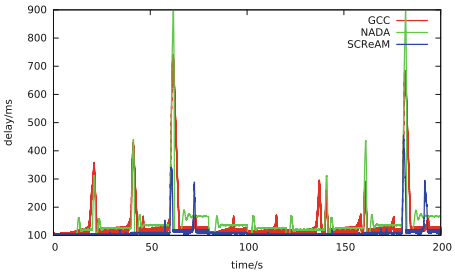

Fig. 2. Packet one way delay

from congestion. NADA can quickly respond to network change. SCReAM is sensitive to capacity decrease, but reacts sluggishly to capacity increase.

The average link bandwidth utilization is shown in Table 1. The link utilization is computed according Eq. (12), in which the term $\overline{x}$ is the average simulated encoder video packets generating rate determined by congestion algorithm and BW is the link capacity during the test period. NADA has the highest channel utilization and SCReAM makes the lowest channel utilization which may cause by its rate ramp-up parameter.

$$bw\_u = \frac{\overline{x}}{BW} \tag{12}$$

Table 1. Average link utilization

Utilization \ Time(s) / Protocol	0-20	20-40	40-60	60-80	80-100
GCC	56.79%	88.10%	89.28%	86.19%	71.58%
NADA	80.41%	95.54%	95.80%	98.69%	92.65%
SCReAM	43.41%	61.08%	87.54%	62.79%	76.76%

From the one-way delay variation curve in Fig. 2, SCReAM reaches its claimed goal by having the lowest queue delay occupation close to one-way link transmission delay. NADA and GCC make link queue build up to some extent. All three protocols show instantaneous delay spike when faced sharp bandwidth decrease.

3.2 Intra Protocol Fairness

Protocol fairness is an important indication to reflect whether an end user converges to a fair bandwidth rate when sharing link with other flows. In this experiment, three flows exploiting the same congestion control protocol were initiated at different time point over a bottleneck link. The second flow was started after 40 s and the third flow was started at 80 s. The link capacity keeps to be a constant value 2 Mbps during the simulation.

In Fig. 3, the rates of all three GCC flows after 150 s are very close, indicating the GCC protocol has fine fairness property. It's worth noticing the NADA protocol suffers from "late-comer effect" in Fig. 4, the late coming flow data sending rate is higher than the flows initiated before. This result is different from the conclusion in [10]. The "late-comer effect" may be caused by that not all flows have equal aggregate congestion price in gradual rate update phase. The SCReAM protocol in Fig. 5 shows no sign that the flows converge to a fairness rate. Due to the effect of link queue building up, the rate adjustment of SCReAM shows oscillation.

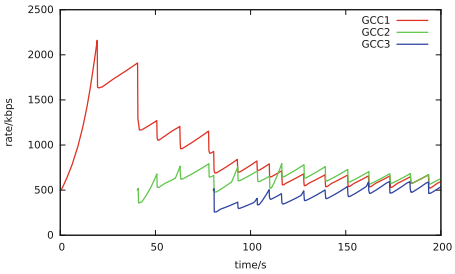

Fig. 3. Sending rate of GCC flows

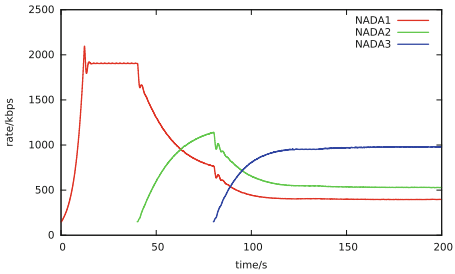

Fig. 4. Sending rate of NADA flows

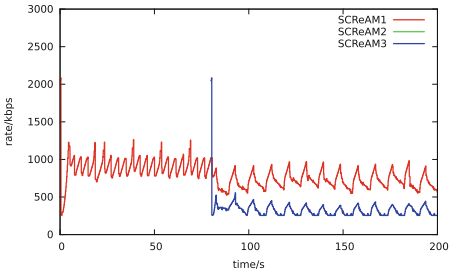

Fig. 5. Sending rate of SCReAM flows

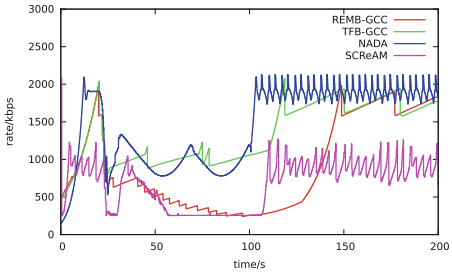

Fig. 6. RMCAT flow sharing links with TCP

3.3 Inter Protocol Competition

In real network, a routing path can be shared by many flows, which may exploit different congestion control protocol. When sharing links with background traffic, the ability to make a reasonable bandwidth occupation of a protocol is quite important. For testing purpose, an experiment was designed for a RMCAT flow sharing link with a TCP Reno flow. The TCP flow was started at 20 s and stopped at 100 s. Even though the REMB-GCC was deprecated in new version of WebRTC, we test its performance here. The result is shown in Fig. 6.

When the TCP connection flows into the link, REMB-GCC flow keeps yielding its bandwidth until reaching the smallest point. TFB-GCC and NADA in Fig. 6 can maintain a reasonable sending rate. SCReAM also decreases its rate to the minimal default rate due to link delay increase caused by the loss based rate control TCP flow. When the link buffer on the merge of full, packet loss event would happen and the TCP flows would half its congestion window to relieve the link from further congestion, the queue delay decrease signal would make NADA and TFB-GCC increase its rate. This explains why the rate curves of NADA and TFB-GCC have increase tendency even in the presence of TCP flow. When TCP flow exits off the network at the time point 100, NADA can make faster increase to reach a rate near the link capacity than TFB-GCC. It should be pointed out NADA flow shows small oscillation even when the tcp flow withdraws from the

link, which is caused by its piecewise rate control function. This is caused by its rate control function in gradual rate update phase for congestion avoidance.

3.4 Packet Loss Resistance

In wireless links, packet loss may cause by wireless link interference, channel contention and errors. A protocol takes random packet loss as congestion signal and reacts it by rate decreasing will have degenerated performance and low channel utilization. In this experiment, the link is configured with different random packet loss rate, and the link is monopolized by a single flow during the simulation.

In experiment, GCC flow is not quite affected by random packet loss. As the packet loss rate increases, NADA and SCReAM decrease bitrate quite obvious. In the case of 5% packet loss, GCC can hold 82.05% channel utilization on average, and both NADA and SCReAM have quite low link utilization shown in Table 2.

Table 2. Capacity utilization in lossy link

Utilization / loss rate / Protocol	0.0%	1%	5%
GCC	86.32%	85.81%	82.05%
NADA	94.28%	92.65%	14.40%
SCReAM	57.62%	15.30%	13.04%

4 Conclusion

The main work of this paper makes a full comparison and analysis of these congestion control algorithms for RTP media in respect of protocol responsiveness, intra protocol fairness, inter protocol competence and performance in loss link.

The results from simulation are summarized here. GCC works well in intra protocol fairness but has saw tooth feature in dynamic links. NADA can quickly stabilize its rate in dynamic links and has the most efficient network capacity utilization when the link is not affected by random loss, but suffers from “late comer effect”. SCReAM retains the link queue delay in a low level but has low channel utilization. GCC has better performance in loss link, which makes it particularly suitable for wireless network. To design a protocol with advantages of these algorithms should be our future work.

With the popular of video based applications, to design new congestion control algorithm for real time traffic will further draw researchers' attention. And the old tree of congestion control research areas always springs new sprouts such as TCP BBR.

References

1. Jacobson, V.: Congestion avoidance and control. ACM SIGCOMM Comput. Commun. Rev. **18**, 314–329 (1988)
2. Xu, L., Harfoush, K., Rhee, I.: Binary increase congestion control (BIC) for fast long-distance networks. In: Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2004, vol. 4, pp. 2514–2524. IEEE (2004)
3. Ha, S., Rhee, I., Xu, L.: CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Oper. Syst. Rev. **42**(5), 64–74 (2008)
4. Widmer, J., Denda, R., Mauve, M.: A survey on TCP-friendly congestion control. IEEE Netw. **15**(3), 28–37 (2001)
5. Padhye, J., Kurose, J., Towsley, D., Koodli, R.: A model based TCP-friendly rate control protocol. In: Proceedings of NOSSDAV 1999 (1999)
6. Rescorla, E.: Congestion control requirements for RMCAT. https://tools.ietf.org/html/draft-ietf-rmcat-cc-requirements-09
7. Lundin, H., Holmer, S., Alvestrand, H.: A Google congestion control algorithm for real-time communication on the world wide web. https://tools.ietf.org/html/draft-ietf-rmcat-cc-requirements-09
8. Zhu, X., et al.: NADA: a unified congestion control scheme for real-time media. https://tools.ietf.org/html/draft-ietf-rmcat-nada-07
9. Johansson, I., Sarker, Z.: Self-clocked rate adaptation for multimedia, RFC 8298. https://www.rfc-editor.org/rfc/rfc8298.txt
10. Carlucci, G., De Cicco, L., Holmer, S., Mascolo, S.: Analysis and design of the Google congestion control for web real-time communication (WebRTC). In: Proceedings of the 7th International Conference on Multimedia Systems, pp. 1–13. ACM (2016)
11. Holmer, S., Flodman, M., Sprang, E.: RTP extensions for transport-wide congestion control. https://tools.ietf.org/html/draft-holmer-rmcat-transport-wide-cc-extensions-01
12. Carlucci, G., et al.: Congestion control for web real-time communication. IEEE/ACM Trans. Network. (TON) **25**(5), 2629–2642 (2017)
13. De Cicco, L., Carlucci, G., Mascolo, S.: Understanding the dynamic behaviour of the Google congestion control for RTCWeb. In: 2013 20th International Packet Video Workshop (PV), pp. 1–8. IEEE (2013)
14. Carlucci, G., De Cicco, L., Ilharco, C., Mascolo, S.: Congestion control for real-time communications: a comparison between NADA and GCC. In: 2016 24th Mediterranean Conference on Control and Automation (MED), pp. 575–580. IEEE (2016)
15. Sarker, Z., Singh, V., Zhu, X., Ramalho, M.: Test cases for evaluating RMCAT proposals. https://tools.ietf.org/html/draft-ietf-rmcat-eval-test-05