Network Performance Analysis of WebRTC

Haoran Fang
University of Colorado at Boulder
Boulder, Colorado
hafa8098@colorado.edu

ABSTRACT

This paper presents a way to test the performance of WebRTC on endpoints. WebRTC nowadays has been widely used in different applications and deployed on lots of web browsers including Google Chrome and Firefox browser. Mobile versions are also under development. running real-time applications may lead to scenarios regarding performance and usability. Congestion control mechanisms will influence the way data is sent and the other performance metrics like delays or losses which are quite important for network communication nowadays. Some mechanisms which have been used in other technologies are used in browsers to handle the internals of WebRTC technology. In this paper, we mainly focus on the performance of the Receive-side Real-time Congestion Control (RRTCC). We use different network scenarios like varying throughput, losses and delay. The result shows that RRTCC is performant when by itself, but starves when the scenario gets worse at some point or when multiple scenarios are combined.

Keywords

WebRTC; RRTCC; network performance; real time communication

1. INTRODUCTION

Video communication is rapidly changing, the dramatical increase of Internet communication between people is forcing technology to support mobile and real-time video experiences in different ways. Applications such as Vime and Skype provide media and real-time communication over the Internet.

Around 2000, a new approach to web browsing had started to develop which was standardized later as the XMLHttpRequest(XHR) API. It enabled web developers to create web applications that didn?t need to navigate to a new page to update their content or user interface. It allowed them to utilize server-based web services that provided access to structured data and other content. Now the web is undergoing yet another transformation that enables individual web browsers to stream data directly to each other without the need for sending it via intermediary servers. This new form of peer-to-peer communication is built upon?a new set of APIs that is being standardized by the Web Real-Time Communications Working Group available at http://www.w3.org/2011/04/webrtc/ of the World Wide Web Consortium (W3C) [8]

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

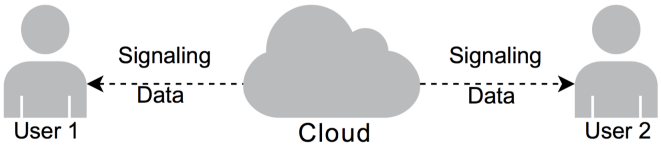

Figure 1: Real time communication between two users over the Internet[7]

1.1 Real-time Communication

Real-time communication is a method of communication among users to exchange information. Latency is important to achieve good quality of communication. Figure 1 shows an RTC scenario for two users, the technology providing the communication may differ in each situation but the goal is always the same.[7]

1.2 WebRTC

WebRTC is part of the HTML5 proposal. It enables RTC capabilities between Internet browsers by using JavaScript APIs which provides video, audio and data transformation. WebRTC provides interoperability among different browsers including Google Chrome, Mozilla Firefox and Opera. With WebRTC, developers can provide applications for most of the desktop devices available, mobile devices will integrate WebRTC as part of their HTML5 package to also enable RTC soon. [1]

WebRTC applications use the GetUserMedia API to allow access to media streams from local devices. It allows developers to access local media devices using JavaScript and generate media streams to be used either with the rest of the *PeerConnection* API or with the HTML5 video element for playback purposes.[4]

2. RECEIVE-SIDE REAL-TIME CONGESTION CONTROL

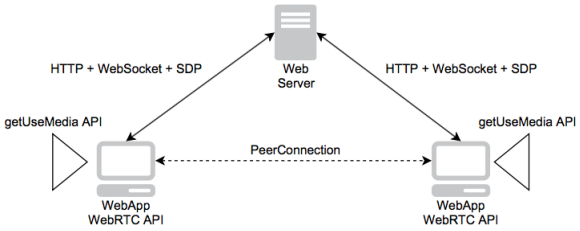

Figure 2: WebRTC simple topology for P2P communication[7]

Real-time Transport Protocol (RTP)[12] carries media data over the network, typically using UDP. In WebRTC, the media packets (in RTP) and the feedback packets (in RTCP) are multiplexed on the same port [10]to reduce the number of NAT bindings and Interactive Connectivity Establishment (ICE) overhead. Furthermore, multiple media sources (e.g, video and audio) are also multiplexed together and sent/received on the same port [11]. In topologies with multiple participants [14]. Media streams from all participants are also multiplexed together and received on the same port [6].

This protocol enables congestion control from the receiver side. The receiver uses an inter-arrival time (IAT) of the packet to calculate the usage of the link. There different states will be described in a link: overuse, stable and underuse. The current receiver estimate can be calculated as follows:

$$A_r(i) = \begin{cases} 0.85 \times RR\ overuse \\ A_r(i-1)\ stable \\ 1.5 \times RR\ under-use \end{cases} \quad (1)$$

The receiving endpoint sends an RTCP feedback message to the sender contain- ing the Receiver Estimated Media Bitrate (REMB)[2]. Once the feedback message is received, the sender side uses the TCP Friendly Rate Control (TFRC) mechanism to estimate the sending rate based on the loss rate, RTT and bytes sent[5]. We can see from this equation that the rate should be stable from the previous state when the loss rate is between 2% to 10%. The new sender available bandwidth (As) is calculated using Equation 9, p being the packet loss ratio. The new sender available bandwidth (A_s) is calculated using equation (2). P stands for the loss rate of packets.

$$A_s(i) = \begin{cases} A_s(i-1) \times (1-0.5p) & p > 0.10 \\ A_s(i-1) & 0.02 \leq p \leq 0.1 \\ 1.05 \times (A_s(i-1)+1000) & p < 0.02 \end{cases} \quad (2)$$

We can see from this equation that this algorithm does not react to losses under 10% and increases the rate by 5% when having losses between 2% and 10%. Forward Error Correction (FEC) mechanism is also included in WebRTC which use FEC packets to protect the stream from losses.

3. EVALUATION ENVIRONMENT

3.1 WebRTC clients

For this project, we use two machines to establish a real-time connection between two end hosts. One is Ubuntu OS and the other one is Ubuntu VM on macbook. Both devices are equipped with webcams.

Table 1: Two Clients

OS	Memory	Processor
Ubuntu	3.7 GB	Intel Core i3 CPU 2.27GHz * 4 64bit
Ubuntu VM	990.5 MB	Intel Core i5 CPU 2.60GHz * 2 64bit

Audio is test has been disabled because of a bug in the *Pulse Audio* module.

apprtc.appspot.com is a webrtc demo application hosted on App Engine. The development AppRTC server can be accessed by visiting http://localhost:8080. Running AppRTC locally requires the Google App Engine SDK for Python and Grunt. Note that logging is automatically enabled when running on Google App Engine using an implicit service account. Figure 3 shows an example of using apprtc.appspot.com.

3.2 Stats API

WebRTC Stats API is a mechanism to get many different kinds of data of a WebRTC application. This technique can be used for debugging. You can get access to some hidden data that is nor visible for customers. Using this part of API you can better understand what is going on under the hood of the web browser and your application.

Since WebRTC is still under development, the API functions might still have different names in the supported web browsers. To solve this issue, it is worthwhile to write additional code that could serve as a wrapper and universal API to the function.[13]

We use *RTCStatsCallback* from the Stats API to return an object which is in JSON format. This object returns as many arrays as streams available in a PeerConnection. This data is provided by the lower layers of the network channel extracting the information from the RTCP packets that come multiplexed in the same network port [9].

RTCStatsCallback allows developers to access different metrics. Methods involved in the RTCStatsCallback are available on the W3C editors draft [3].

A high level Statas API has been built to calculate the RTT, throughput and loss rate for different traffic streams that are sent over the PeerConnection. Those stats can be saved after each call. We use JavaScript API to grab any PeerConnection passed through the variable and starts several iterations to collect those data and then plot them out.

3.3 Webrtc-Internal

WebRTC-internals is a built-in mechanism in Chrome with the use of which you can get access to a variety of WebRTC stack-related information and statistics data. Along with the Stats API, we can use Webrtc-Internal to generate graphs. Figure 4 shows the Internal UI.

3.4 Iproute2 tools

iproute2 is a suite of command line utilities which manipulate kernel structures for IP networking configuration on a machine. The binary tc is the only one used for traffic control. disc scheduler is also used to manipulate traffic. This these utility, we can add bandwidth limitations, delays, packet losses and other distortions on the ongoing link.

4. PERFORMANCE EVALUATION

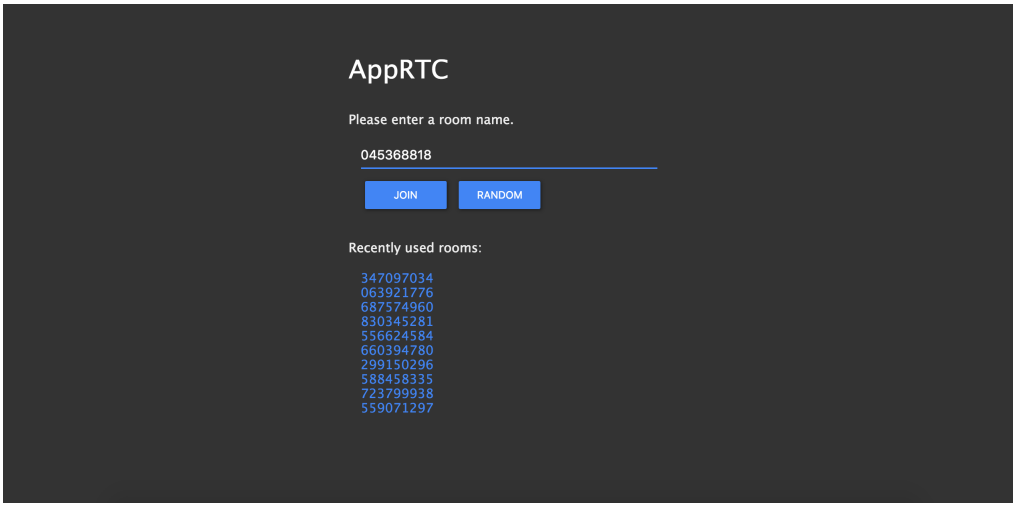

Figure 3: A sample black and white graphic that needs to span two columns of text.

Figure 4: UI for Chrome Internals

For this project, we basically use a single RTP WebRTC flow with different networks constraints. We run each call for 10 mins and 5 times to derive statistical significance. We use the following metrics to measure the performance.

- **Throughput:** the receiver bit rate
- **Loss:** Packet loss is the failure of one or more transmitted packets to arrive at their destination. This event can cause noticeable effects in all types of digital communications.
- **Residual Loss:** the number of packets lost after retransmission or applying Forward Error Correction
- **Delay:** The delay of a network specifies how long it takes for a bit of data to travel across the network from one node or endpoint to another. It is typically measured as RTT.
- **Average Bandwidth Utilization (ABU):** the ratio of the sending or receiving rate to the network capacity.

4.1 Delay

Different one-way latencies have been added to the test including 50, 100, 200 and 500ms. We use those latencies to see if there will be decrease of throughput in a communication. Delay modeling for real time applications is difficult, an approach can be done using the remote timestamp of the incoming packets.[7] We use IAT here to calculate the time. A single bi-directional RTP stream flows has been established and we add one-way latency to the bottleneck links.

Table 2: Summary of averaged results with different latency conditions

Delay	Metrics	WebRTCClient
50ms	bitRate(Kbit/s)	1867.23
	RTT(ms)	250
	Residual Loss(%)	0.02
	Packet Loss(%)	0.03
100ms	bitRate(Kbit/s)	1789.56
	RTT(ms)	310
	Residual Loss(%)	0.12
	Packet Loss(%)	0.83
200ms	bitRate(Kbit/s)	1233.96
	RTT(ms)	408
	Residual Loss(%)	0.5
	Packet Loss(%)	0.67
500ms	bitRate(Kbit/s)	423.37
	RTT(ms)	1245.81
	Residual Loss(%)	0.91
	Packet Loss(%)	0.65

We can see from figure 5- 8 and table 2 that throughput starts to drop when the delay is added to 500ms. Besides, what is interesting is that with the increase of network delay, the packet loss doesn?t increase drastically.

4.2 Loss

In this part, we will evaluate the response of WebRTC calls in different lossy scenarios. In real life, those lossy environment can be reflected in mobile environments with poor network coverage or with heavy congestions. Discarding packets in the peers for large delay also produces losses that can affect the call.[7]

If communication in WebRTC encounters heavy losses, the frame rate of the video must degrade. Hence, the user experience of this video conferencing will be affected exponentially. We have tested a bi-directional bottleneck call with 1, 5, 10 and 20% packet loss. The test results are show in table 3 and figure 9 - 12. The table and graphs show that the bitRate drops drastically when the loss goes up to 20%. It can be demonstrated in the RRTCC algorithm which does

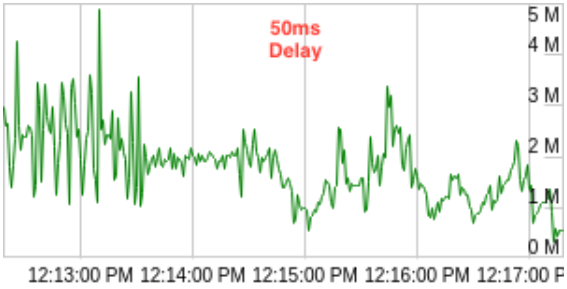

Figure 5: 50ms delay

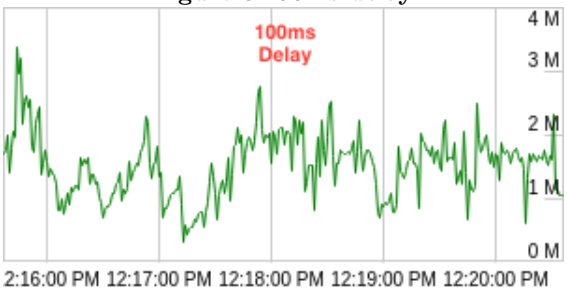

Figure 6: 100ms delay

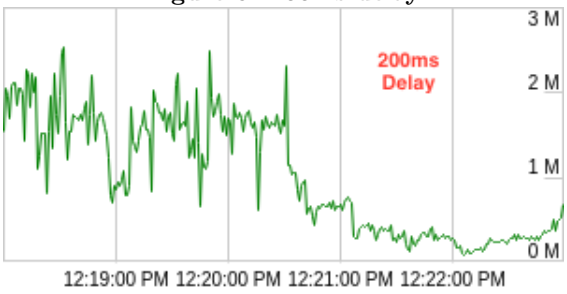

Figure 7: 200ms delay

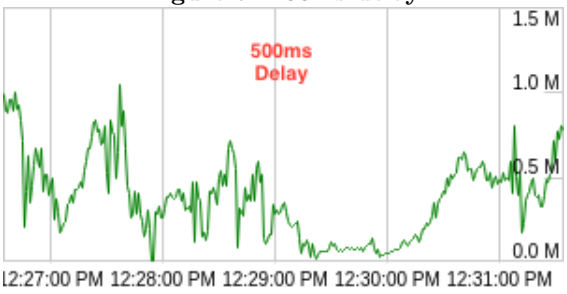

Figure 8: 500ms delay

Table 3: Different Packet Loss Rates

Loss Rate	bitRate(Kbps)	RTT(ms)	Residual Loss
1%	1930.4	13.86	0.089
5%	1945.27	10.2	0.52
10%	1593.51	8.89	0.86
20%	379.42	5.59	3.24

Table 4: Different Queue Sizes

	Throughput: 1Mbps			
QueueSize	Rate(Kbps)	RTT(ms)	Residual Loss(%)	Loss(%)
100ms	452.49	40.33	0.59	1.98
1s	577.87	32.35	0.34	0.05
10s	689.01	30.45	0.12	0.03
100ms	2195.21	30.45	0.02	0.04
1s	1910.49	22.24	0.02	0.03
10s	1420.55	21.39	0.03	0.03

not react to loss under 10%.

WebRTC can negotiate the use of FEC mechanisms between peers. WebRTC may also use redundant transmission as a standard feature of the payload format used in a call. WebRTC FEC mechanism may ask for a retransmission of the lost packet on some occasions, if the delay is small it can be possible to forward them in real time, this metric is then referenced as Residual Loss. [7] Those type of losses will be the ones directly affecting the data received on the peer so a low value will barely affect the quality of the call.[15]

4.3 Bandwidth and queue variations

Next, we vary the queue size to observe the impact on the performance of RRTCC. have selected to run 1Mbps and 5Mbps throughput and different queue lengths.We still use the iproute2 to simulate queue size in router. In practice the queue size is measured in number of packets. We use time metric to describe the queue size by using the following equation.

$$Queue_{packets} = \frac{Bandwidth_{bits}}{8 \times MTU} \times Queue_{seconds} \quad (3)$$

The results of the test is shown in table 4. This result shows that the Average Bandwidth Utilization is almost the same in two conditions. So the queue size in router does not have significant impact on the performance.

4.4 Loss and Latency

Compared with table 3 and table 4, we can observe that RRTCC cannot handle the retransmissions of packets using FEC. The reason is that the residual loss in both tables still increases in all latencies while some of the packet loss remains the same. We test the performance in the combined situation where bandwidth limitation is 1 Mbit/s and queue size is 500ms and loss rate is 10%. Figure 13 shows the result of this test.

This figure shows how how the sending rate reduces when facing loss and delay on the same path RRTCC being unable to fix this issue with its internal mechanisms. I also inspect that the RTT keeps constant during all the duration of the call. The sender decision will be based on the RTT, packet loss and available bandwidth that is estimated

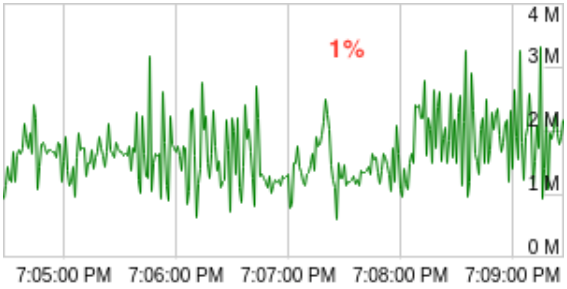

Figure 9: 1% loss

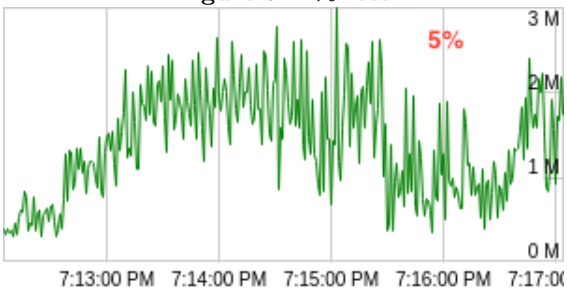

Figure 10: 5% loss

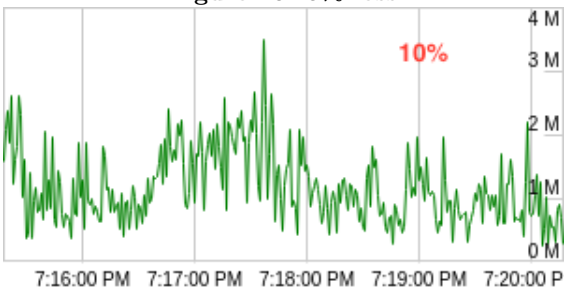

Figure 11: 10% loss

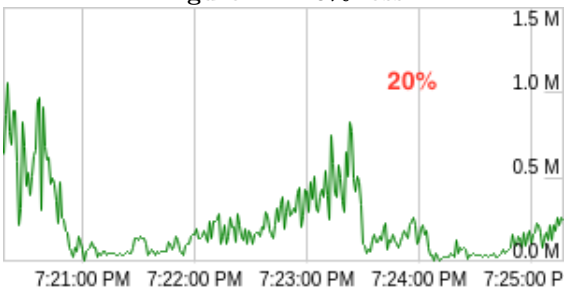

Figure 12: 20% loss

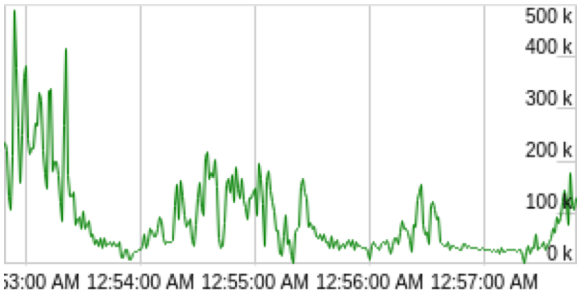

Figure 13: 10% packet loss and 50ms delay in 1Mbit/s limit

from the receiving side using equation (2). However the real output is different from using the formula. The reason is because the sender rate is limited by another protocol called TCP Friendly Rate Control. So if loss and latency happen at the same path, the RRTCC in WebRTC will not produce a better throughput. [7]

5. CONCLUSIONS AND FUTURE WORK

In this project, we tested different scenarios with different network constraint to emulate the real-world situation. First, we use different delays. We show that the RRTCC algorithm is performant in networks with latencies up to 200ms. The bandwidth utilization decreases when the latency exceed 200ms. Second, multiple loss rates are tested which demonstrates that RRTCC algorithm will not react to loss under 10%. We also observe that the residual losses remains low when the packet loss is high. It can be explained that the sender will use retransmission and FEC mechanism to protect the media flow. And the end, we introduce loss and latency in the bottleneck link at the same time. The result may show that RRTCC can no longer rely on retransmissions to handle this situation.

For future works, I may consider different scenarios to test the performance of WebRTC. For example, introducing TCP cross traffic in WebRTC flow or using multiple RTP flows. Besides, I can also try to test the interoperability between different web browsers or test the performance in a mobile environment.

References

[1] S. Alund. Browser- the world first webrtc enabled mobile browser.

[2] H. Alvestrand. Rtcp message for receiver estimated maximum bitrate. *http: //tools.ietf.org/html/draft-alvestrand-rmcat-remb*, 2013.

[3] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan. Webrtc 1.0: Real-time communication between browsers. *http://dev.w3.org/2011/webrtc/editor/webrtc.html*, Jan, 2013.

[4] D. Burnett, A. Bergkvist, C. Jennings, and A. Narayanan. Media capture and streams. *http://dev.w3.org/2011/webrtc/editor/ getusermedia.html*, December 2012.

[5] M. Handley, S. Floyd, J. Padhye, and J. Widmer. Tcp friendly rate con- trol (tfrc): Protocol specification. *http://tools.ietf.org/html/rfc5348*, 2008.

[6] C. Holmberg, S. Hakansson, and G. Eriksson. Rtp topologies. *IETF Internet Draft*.

[7] A. A. Lozano. Performance analysis of topologies for web-based real-time communication. 15.8.2013.

[8] R. Manson. *Getting Started with WebRTC*. Packt Publishing, September 2013.

[9] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. Real-time transport control protocol (rtcp)-based feedback (rtp/avpf). *http://www.ietf. org/rfc/rfc4585.txt*, 2003.

[10] C. Perkins and M. Westerlund. Multiplexing rtp data and control packets on a single port. *Internet Engineering Task Force*, RFC 5761, Apr. 2010.

[11] C. Perkins, M. Westerlund, and J. Ott. Web real-time communication (webrtc): Media transport and use of rtp. *IETF Internet Draft*.

[12] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. *Internet Engineering Task Force*, RFC 3550, July 2003.

[13] A. Sergiienko. *WebRTC Cookbook*. Packt Publishing, 2013.

[14] M. Westerlund and S. Wenger. Rtp topologies. *IETF Internet Draft*, April 2013.

[15] X. Yu, J. W. Modestino, and D. Fan. Evaluation of the residual packet-loss rate using packet-level fec for delay-constrained media network transport. *http://info.iet.unipi.it/ luigi/papers/ 20091201-dummynet.pdf*, November 2006.