International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 1 | ISSN : 2456-3307

Web Real-Time Communication by Controlling Congestion

D. Prakash Rao[*1], Dr. G. S.Bapi Raju[2]
[*1]M.Tech, CSE, Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad, Telangana, India
[2]Professor, CSE, Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad, Telangana, India

ABSTRACT

WebRTC has rapidly turned out to be famous as a video conferencing stage, halfway because of the way that numerous programs support it. WebRTC uses the Google Congestion Control (GCC) calculation to give clog control to ongoing correspondences over UDP. The execution amid a WebRTC call might be affected by a few variables, including the fundamental WebRTC usage, the gadget and system qualities, and the system topology. In this paper, we play out an intensive execution assessment of WebRTC both in copied manufactured system conditions and in genuine wired and remote systems. The assessment demonstrates that WebRTC streams have a marginally higher need than TCP streams while rivaling cross activity. When all is said in done, while in a few of the considered situations WebRTC Performed obviously, we watched essential situations where there is opportunity to get better. These incorporate the remote area and the recently included help for the video codec's VP9 and H.264 that does not execute of course.

Keywords: WebRTC, Congestion Control, Performance Evaluation.

I. INTRODUCTION

WebRTC gives Real-Time Communication (RTC) capacities by means of program to-program correspondence for sound (voice calling), video talk, and information (_le sharing). It enables programs to discuss specifically with each other in a peer-to-peer fashion, which contrasts from regular program to web-server correspondence. One of the fundamental points of interest of WebRTC is that it is coordinated in most present day web programs and keeps running without the need to introduce outside modules or applications. The World Wide Web Consortium (W3C) [4] has set up an Application Programming Interface (API), which enables designers to effortlessly actualize WebRTC utilizing JavaScript, while the Internet Engineering Task Force (IETF) [14] characterizes the WebRTC conventions and basic organizations.

To understand the low dormancy and high throughput fundamental IFIP WG 7.3 Performance 2017. Nov. 1416, 2017, New York, NY USA Copyright is held by creator/owner(s). For constant correspondence, WebRTC organizes transmit-ting information utilizing UDP rather than TCP. WebRTC over TCP is utilized if all else fails, when all UDP ports are blocked, which can be the situation in intensely secured undertaking net-works. Since UDP does not bolster any type of blockage control, WebRTC utilizes a hand crafted clog control calculation that adjusts to changing system conditions. With the abnormal state API, WebRTC makes it simple for application engineers to build up their own particular video spilling applications. The impediment of this abnormal state approach is that the execution points of interest, particularly the way conges-tion is dealt with, are totally avoided application designers. In the meantime, late research assessing the execution of WebRTC has just somewhat tended to this hole (see Section 7 for more subtle elements). In this paper, we investigate the execution of WebRTC, chiefly concentrating on the Google Congestion Control (GCC) calculation, which is the most broadly utilized blockage control calculation for WebRTC. We assess its execution utilizing the most recent web programs over an extensive variety of utilization cases. The key commitments comprise of concentrate the impacts of various engineered arrange conditions on the most recent usage of WebRTC, looking at WebRTC's execution on cell phones, dissecting the execution of the recently included video

codec's VP9 and H.264, and assessing the effect of wired and remote systems on WebRTC. The source code for replicating the test conditions depicted in this paper is accessible at: https://gitnode.com/Wimnet/webrtc_performance specifically, my trial examine incorporates the accompanying:

Benchmark Experiments: I examine the impacts of shifting dormancy, parcel misfortune, and accessible data transfer capacity by imitating diverse execution situations utilizing Dummy net. We set up benchmarks for the execution of WebRTC in various situations.

Cross Traffic: I consider the impacts of TCP cross activity and various WebRTC streams having a similar bottleneck. The assessments demonstrate that with the re-penny improvements to the clog control system, WebRTC streams get marginally higher need while contending with TCP streams.

Multi-Party Topology: We analyze the execution of a work and Selective Forwarding Unit (SFU) based topologies for gather video calls utilizing WebRTC. The assessment features inborn exchange o s amongst execution and sending extra foundation for multi-party video calls.

Video Codec's: We contemplate the execution of three generally utilized video codec's, VP8, VP9, and H.264, on WebRTC. My investigations exhibit that the recently included H.264 and VP9 codec's don't execute not surprisingly within the sight of clog or bundle misfortunes.

Versatile Performance: We assess the execution of WebRTC on cell phones and exhibit the effect of restricted computational limit available to come back to work quality.

Genuine Wireless Networks: We tentatively assess video approaches WebRTC in genuine systems, particularly concentrating on remote systems. My trials demonstrate that WebRTC can experience the ill effects of poor execution over remote because of burst misfortunes and parcel retrains-missions. We recognize key regions for development and brie y take a gander at cross-layer approaches for enhancing video quality. Here Performance assessment and plan of clog control calculations for live video spilling have gotten consider-capable consideration. Underneath, we feature the most important work.

Blockage control for sight and sound: TCP variations, for example, Tahoe and Reno [16] have appeared to prompt poor execution for mixed media applications since they depend just on misfortunes for clog sign. The ways to deal with address the deficiencies of these strategies can be separated in two classifications.

The main assortment of blockage control calculations utilizes variations of postponement to deduce clog. Postpone based variations of TCP, for example, Vegas [5], and FAST [24] depend on measuring round trek delays yet they are more receptive than proactive in clog control. LEDBAT [22] depends on measuring one way parcel postponements to guarantee high throughput while limiting deferrals. Grow [25] uses stochastic conjectures of cell arrange execution to accomplish similar objectives. The second classification of blockage control depends on Active Queue Management (AQM) systems. Nothing [27] utilizes Explicit Congestion Notifications (ECN) and misfortune rate to get an exact gauge of misfortunes for clog control.

WebRTC clog control: SCReAM [17] is a cross breed misfortune and postpone based blockage control calculation for conversational video over LTE. FBRA [19] proposes a FEC-based clog control calculation that tests for the accessible transmission capacity through FEC bundles. On account of misfortunes because of blockage, the excess bundles help in recouping the lost parcels.

WebRTC execution assessment: Several papers have examined the execution of WebRTC. Most related work concentrates on a solitary part of the convention or utilize obsolete adaptations of WebRTC in their execution investigations. [2] Analyzes the Janus WebRTC portal concentrating on its execution and versatility just for sound conferencing in multi-party calls. [8] Focuses on examination of end-to-end and AQM-based clog control calculations. [7] Evaluates the execution of WebRTC over IEEE 802.11 and proposes methods for gathering bundles together to keep away from GCC's activity on bursty misfortunes.

[10] Presents the plan of the latest variant of the GCC calculation utilized as a part of the WebRTC stack. While [10] expert vides preparatory examination of GCC in some engineered organize conditions, it doesn't concentrate on WebRTC's execution on cell phones or genuine wired and remote systems. Its primary concentrate is on between convention reasonableness between various RTP streams and RTP streams contending with TCP takes after.

[23] gives an imitating based execution assessment of WebRTC. Be that as it may, all aws identi ed in [23] have been in this way tended to in WebRTC. For example, the information rate never again drops at high latencies (however rather reacts to inertness variety), the data transmission sharing amongst TCP and RTP is more attractive because of the recently presented dynamic limit, and the accessible transfer speed is shared all the more similarly while contending RTP ows are included.

A more sensible execution ponder utilizing genuine system e ects is done in [13], where the execution of WebRTC is measured with portable clients in di erent territories. Despite the fact that the WebRTC execution utilized is obsolete, the paper proposes that WebRTC's over-dependence on bundle misfortune signals prompts under-usage of the channel because of portability.

System Architecture

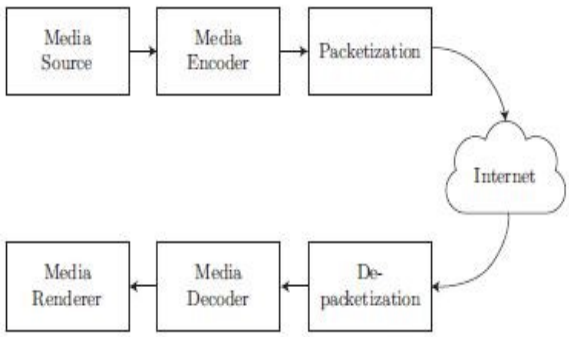

Figure 1. WebRTC's media processing pipeline.

II. IMPLEMENTATION

WebRTC utilizes the Google Congestion Control (GCC) calculation [15], which progressively modifies the information rate of the video streams when clog is identified. In this area, they give a short outline of GCC. More points of interest can be found in [10].

WebRTC normally utilizes UDP (unless all UDP ports are obstructed), over which it utilizes the Real-time Trans-port Protocol (RTP) to send media bundles. It gets criticism bundles from the collector as RTP Control Protocol (RTCP) reports. GCC controls blockage in two ways: delay-based control at the less than desirable end and misfortune based control at the sender side.

Receiver-side controller

The collector side controller is delay-based and thinks about the timestamps of the got outlines with the time moments of the edges' age. The collector side controller comprises of three unique subsystems: (I) entry time modify, (ii) over-utilize indicator, and (iii) rate controller. These distinctive subsystems of the collector side controller are appeared on the correct side of Figure 1. The entry time adjust (Section evaluates the progressions in lining deferral to identify clog. The over-utilize identifier recognizes the clog by looking at the assessed lining postpone changes from the entry time adjust with a versatile edge. The rate controller settles on the choices to build, diminishing, or hold the evaluated accessible rate at the recipient, Ar, in light of the blockage assessed got from the over-utilize locator. Ar(i) for the ith video outline is given as takes after:

$$Ar(i) = \begin{cases} \eta Ar(i-1) & \text{Increase} \\ \alpha R(i) & \text{Decrease} \\ Ar(i-1) & \text{Hold} \end{cases}$$

Where = 1:05, = 0:85, and R(i) is the measured received rate for the last 500 ms. The received rate can never exceed 1:5R(i):

$$A_r(i) = \min(A_r(i); 1{:}5R(i))$$

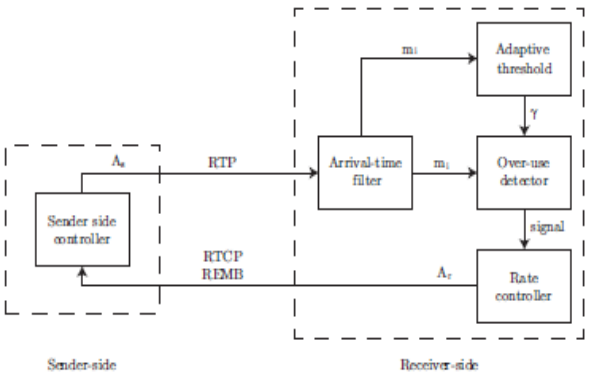

Figure 2. Diagram illustrating how sender and receiver determine and exchange their available rate.

Arrival-time filter

The entry time channel constantly measures the time moments at which parcels are gotten. It utilizes the season of entries to ascertain the between landing time between two back to back bundles: ti ti 1, and the between flight time between the transmission of similar parcels: Ti Ti 1. It at that point computes the restricted postpone variety di, characterized as the contrast between entry time and between flight time as takes after:

$$di = (ti \quad ti\ 1)\ (Ti \quad Ti\ 1)$$

This postpone demonstrates the relative increment or diminishing concerning the past parcel. The restricted defer variety is bigger than 0 if the between landing time is bigger than the between takeoff time. The entry time changes evaluates the restricted lining defer variety mi. The computation of mi depends on the deliberate di and past state gauge mi 1, whose weights are powerfully balanced by a Kalman adjust to decrease commotion in estimation. For example, the weight for the present estimation di is measured more vigorously than the past gauge mi 1 when the blunder difference is low. For more subtle elements, see [15].

Over-use detector

The assessed one-way lining defer variety (mi) is contrasted with a limit. Over-utilize is recognized, if the gauge is bigger than this limit. The over-utilize indicator does not flag this to the rate controller, unless over-utilize is identified for a predetermined timeframe. The over-utilize time is as of now set to 100ms [10]. Under-utilize is distinguished when the gauge is littler than the negative estimation of this limit and works in a comparative way. An ordinary flag is activated when mi.

The estimation of the limit largy affects the general execution of the GCC blockage calculation. A static edge can undoubtedly bring about starvation within the sight of simultaneous TCP streams, as appeared in [11]. Thusly, a dynamic limit was executed as takes after:

$$^{\gamma}_{i} = {}^{\gamma}_{i-1} + (t_i - t_i - {}_1)\ *K_i *\ (|m_i\ {}^{\gamma}_{i}\ {}_{-1})$$

The value of the gain, K_i, depends on whether jm_ij is larger or smaller than $_{i\ 1}$:

$$K_i = \begin{cases} K_d & |m_i| < {}^{\gamma}_{i-1} \\ K_u & \text{otherwise} \end{cases}$$

Where Kd < Ku. This makes the limit increment when the evaluated mi isn't in the scope of [I 1; I 1] and diminish when it falls in that range. This helps expanding the limit when, e.g., a simultaneous TCP takes after enters the bottleneck and keeps away from starvation of the WebRTC streams. As per [11], this versatile limit brings about 33% better information rates and 16% lower RTTs when there is contending activity having a similar bottleneck.

Rate controller

The rate controller chooses whether to expand, lessening, or hold Ar at the recipient relying upon the flag got from the over-utilize identifier. At first, the rate controller continues expanding Ar until over-utilize is distinguished by the over-utilize locator. Figure additionally delineates how the rate controller modifies in view of the signs got by the over-utilize identifier.

A clog/over-utilize flag dependably brings about diminishing the rate, while under-utilize dependably brings about keeping the rate unaltered. The condition of the rate controller converts into accessible rate at the beneficiary, Ar, as appeared in condition (1). Ar is sent back to the sender as a REMB (Receiver Estimated Maximum Bandwidth) 1 message in a RTCP report.

Sender-side controller

The sender-side controller is misfortune based and registers the sending rate at the sender, As in Kbps and is appeared on the left half of Figure 1. As is Figured each time (tk) the kth RTCP report or a REMB message is gotten from the beneficiary. The estimation of As depends on the part of lost bundles fl(tk) as takes after:

$$As(tk) = \begin{cases} As(tk\text{-}1)(1 - 0{:}5fl(tk)) & fl(i) > 0{:}1 \\ 1{:}05As(Tk\text{-}1) & fl(tk) < 0{:}02 \\ As(tk\text{-}1) & \text{otherwise} \end{cases}$$

On the off chance that the bundle misfortune is in the vicinity of 2% and 10%, the sending rate stays unaltered. On the off chance that over 10% of the bundles is accounted for lost, the rate is multiplicatively diminished. In the event that the parcel misfortune is littler than 2%, the sending rate is directly expanded. Besides, the sending rate can never surpass the last accessible rate at the beneficiary Ar(tk), which is gotten through REMB

messages from the recipient as found in Figure1.

Experimental Setup

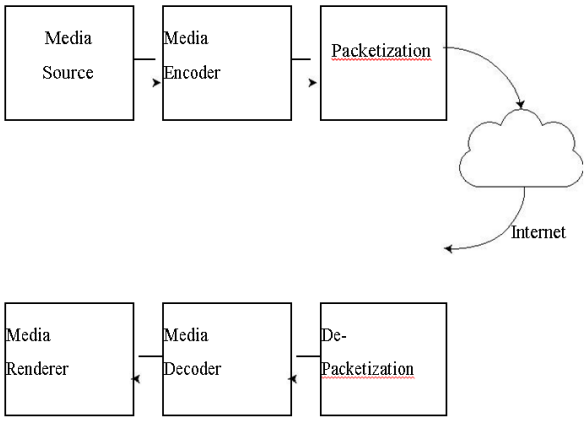

Figure 3. WebRTC's media processing pipeline.

In this area, the portray setup utilized for experimental assessment all through the paper. WebRTC handles all media preparing as showed in Figure 3. Raw media from the sound and video source are rst preprocessed and after that encoded at a given target rate. These casings are then packe-tized and sent to the recipient over RTP/UDP. These edges are therefore depacketized and decoded, which gives the crude video input that can be rendered at the recipient.

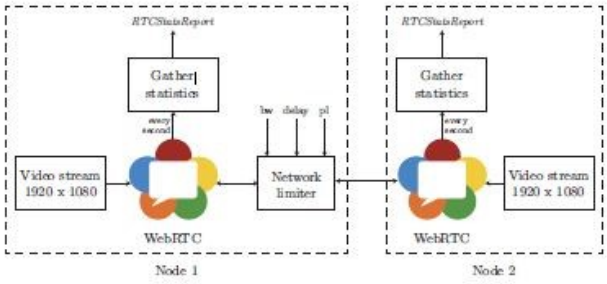

Figure 4. Experimental setup used for performance evaluation where the network limiter is simulated using Dummynet.

The assessment of WebRTC is isolated into two sections. In the rest part, the imitate engineered arrange conditions to examine the execution of WebRTC in controlled settings. In the second part, the concentrate on trial assessment on genuine systems and especially concentrate on remote systems. The trial assessment setup for two clients is appeared in Figure 4.

For the rest part, I imitate distinctive system attributes utilizing Dummynet [6], which enables us to include idleness, parcel misfortune, and point of confinement the transfer speed for both uplink and downlink. To maintain a strategic distance from extra inertness and system confinements, the interface both WebRTC endpoints to a similar nearby system through a wire.

In the greater part of our tests, the utilize gadgets with adequate handling and memory ability to guarantee that the encoding and translating of the video streams are not influenced because of the gadgets themselves. To guarantee this, they use WebRTC's RTC Stats Report API usefulness which demonstrates if the video quality is restricted because of memory or calculation control at the gadgets. Unless specified else, I utilize the latest form of WebRTC (bolstered by Google Chrome adaptation 52 and onwards) at all customers, with the default sound and video codec's OPUS and VP8, separately. Rather than utilizing a webcam encourage and amplifier sound flag, the misuse Google Chrome's phony gadget usefulness to bolster the program a circling video and sound track to get similar outcomes. For every one of our tests (unless specified else), I utilize the accompanying video with a determination of 1920x1080 at 50 outlines for each second with consistent bitrate: in to tree 2. To get execution measurements, the utilize WebRTC's worked in RTCStatsReport 3, which contains point by point insights about information being exchanged between the companions.

III. SYNTHETIC NETWORK CONDITIONS

In this section I evaluate the performance of WebRTC's GCC algorithm in synthetic yet typical network scenarios using Dummynet.

Static network conditions

Figure 5 demonstrates the outcomes for the situations when both the uplink and downlink transfer speed are constrained to 1500Kbps, 750Kbps, and 250Kbps. See that WebRTC is dog as of late constrained to sending at 2500Kbps, as set in the program 4. At the point when the transfer speed is restricted, it utilizes 80% of the accessible transmission capacity and can keep up a steady information rate. By persistently bringing down the accessible data transmission in extra tests, I watched that at least 20Kbps is important to build up a video call between two gatherings. In any case, no less than 250Kbps of accessible transmission capacity is important to get a to some degree worthy casing rate (25 Frames for every Second (FPS)) at the most reduced conceivable

video determination (480x270). It takes longer to achieve the greatest information rate, particularly when I take a gander at the 250Kbps, where it takes around 10 seconds for any information to take after between the two nodes.

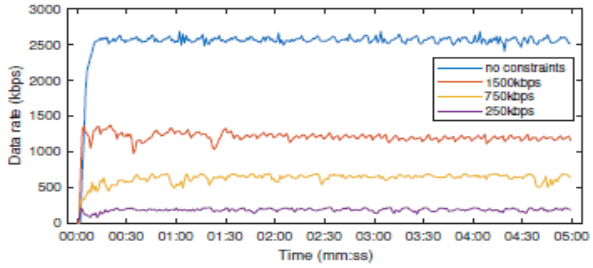

Figure 5. Data rate with limited bandwidth and without any constraints (100Mbps or more available bandwidth).

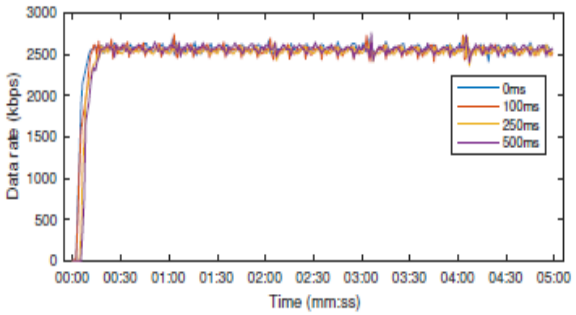

Figure 6. Data rate with additional latency.

Next, they add additional inactivity to the call, as appeared in Figure 6. Obviously, this does not influence the information rate, since the GCC calculation just reacts to idleness variety. How-ever, it prompts delays in the discussion. ITU-T Recommendation G.114 [1] determines that restricted transmission deferral ought to ideally be kept beneath 150ms, and delays over 400ms are viewed as unsatisfactory. While including delay, additionally watch that it takes more time to set up the call and for information to stream between both end focuses, which contrarily influence client encounter. When information streams, it takes around 10 seconds to achieve its greatest information rate, paying little mind to the additional postponement. This deferral is not as much as what is normal from the GCC conditions where the rate would increment with 5% as appeared in condition (6). This is on the grounds that once an association is set up, WebRTC utilizes a conceivable.

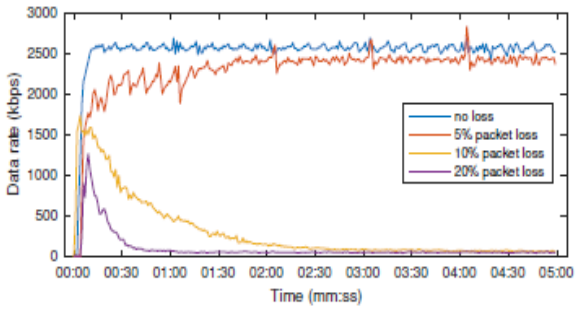

Figure 7. Data rate with packet loss.

I for the following examination drop a specific level of all bundles: 5%, 10%, and 20%. The outcomes are appeared in Figure 7. The outcomes coordinate our desires in light of Equation (6). GCC just reductions the sending rate when over 10% bundle misfortune is distinguished. The sending rate stays unaltered in the vicinity of 2% and 10% and the rate is expanded when under 2% of the parcels are lost. There-fore, 5% parcel misfortune gradually merges to the most extreme information rate and at 10% bundle misfortune; the information rate joins to at least 50Kbps, which totally comprises of sound information (the sound stream isn't liable to clog control by GCC because of its low information rate [12]).

IV. WIRELESS PERFORMANCE

In this area, I assess the execution of WebRTC over genuine systems. I particularly concentrate on concentrate the effect of a Wi-Fi hop on WebRTC.

Benchmarking

In Section 4, I watched that GCC is delicate to changes in dormancy and parcel misfortunes. Transmitting over remote net-works may bring about burst parcel misfortunes and dynamic latencies because of resulting retransmissions, particularly if the conclusion to-end Round Trip Time (RTT) of the WebRTC association is substantial. In this area, portray the impacts of remote connections on the execution of WebRTC by looking at against the execution on wired connections.

I consider 3 kinds of WebRTC nodes:i) a local wireless node, (ii) a local wired node, and (iii) remote wired nodes. I utilized a 2013 ASUS Nexus 7 tablet as a neighbourhood remote node associated with an IEEE 802.11 DD-WRT empowered Access Point (AP). The wired node is either a neighbourhood machine situated in our lab in New York City or a remote server running

in Amazon EC2 cloud. I think about two cases for the remote server: one in the AWS Oregon accessibility zone and one in the AWS Sydney accessibility zone which give distinctive extents of RTT. This enables us to contemplate the effect of higher RTT when contrasted with the neighbourhood machine.

Both the local and remote machines run Ubuntu 14.04 with Google Chrome 57.0 as the program. I utilize the same infused video les for a reasonable examination. In addition, every one of the machines have adequate computational energy to dispose of the effect of gadgets on video execution. A virtual show support was utilized on the EC2 servers to run WebRTC on Chrome in headless mode. For the remote node, I utilized 5GHz channels to limit the impedance from other IEEE 802.11 systems. To copy the states of high misfortune situations, the AP transmission influence was set to 1mW. We explore different avenues regarding diverse channel conditions with the remote node being in an indistinguishable room from the AP (roughly 5 feet away), and also outside of the room (around 25 feet away).

Figure 7 demonstrates normal call measurements for two completely wired calls with one wired node situated in the NYC territory in the lab and the other node in Oregon or Sydney. The NYC node was infusing a video encoded at 50FPS, and the remote nodes were utilizing a video encoded at 60FPS. The normal RTTs for the Oregon and Sydney calls were 214.86ms, separately. As needs be, the term these situations as \medium" and \high" call latencies when contrasted with \short" inactivity situation with the two nodes in the NYC zone. These outcomes set up a standard execution of WebRTC in practical system conditions.

Next, I perform video calls with one remote node and the other node either being a neighbourhood wired node or one of the two remote nodes. A 720p video encoded in 50FPS was utilized over every one of the 3 cases. On the remote node, the cam-period on the Nexus tablet was utilized as video source, since video couldn't be infused into the Android conveyance of Chrome without establishing the gadget.

V. CONCLUSION

In this paper, I assessed the execution of WebRTC-based video conferencing, with the fundamental concentrate being on the Google Congestion Control (GCC) calculation. Our assessments in manufactured, yet ordinary, arrange situations demonstrate that WebRTC is delicate to varieties in RTT and parcel misfortunes. The likewise assessed the effect of various video codec's, mo-bile gadgets, and topologies on WebRTC video calls. Further, our assessments on genuine wired and remote systems demonstrate that burst bundle misfortunes and retransmissions over long RTTs can particularly prompt poor video execution. The source code for setting up and assessing the trial situations portrayed in this paper is accessible at: https://gitnode.com/Wimnet/webrtc performance.

VI. REFERENCES

[1]. One-way transmission time. ITU-T, G.114 (May 2003).

[2]. Amirante, A., Castaldi, T., Miniero, L., and Romano, S. P. Performance analysis of the janus webrtc gateway. In Proc. ACM AWeS'15 (2015).

[3]. Ammar, D., De Moor, K., Xie, M., Fiedler, M., and Heegaard, P. Video QoE killer and performance statistics in WebRTC-based video communication. In Proc. IEEE ICCE'16 (2016).

[4]. Bergkvist, A., Burnett, D. C., Jennings, C., Narayanan, A., and Aboba, B. Webrtc 1.0: Real-time communication between browsers. online, 2016. http://www.w3.org/TR/webrtc/.

[5]. Brakmo, L. S., and Peterson, L. L. TCP Vegas: End to end congestion avoidance on a global internet. IEEE J. Sel. Areas Commun. 13, 8 (1995), 1465{1480.

[6]. Carbone, M., and Rizzo, L. Dummynet revisited. SIGCOMM Comput. Commun. Rev. 40, 2 (2010), 12{20.

[7]. Carlucci, G., De Cicco, L., Holmer, S., and Mascolo, S. Making Google congestion control robust over Wi-Fi networks using packet grouping. In Proc. ACM ANRW'16 (2016).

[8]. Carlucci, G., De Cicco, L., and Mascolo, S. Controlling queuing delays for real-time communication: the interplay of E2E and AQM algorithms. ACM SIGCOMM Computer Commun. Rev. 46, 3 (2016).

[9]. Chen, W., Ma, L., and Shen, C.-C. Congestion-aware MAC layer adaptation to improve video telephony over Wi-Fi. ACM Trans. Multimedia Comput. Commun. Appl. 12, 5s (2016), 83:1{83:24.

[10]. Cicco, L. D., Carlucci, G., Holmer, S., and Mascolo, S. Analysis and design of the google congestion control for web real-time communication (WebRTC). In Proc. ACM MMsys'16 (2016).

[11]. Cicco, L. D., Carlucci, G., and Mascolo, S. Understanding the dynamic behaviour of the google congestion control for RTCWeb. In Proc. IEEE PV'13 (2013).

[12]. De Cicco, L., Carlucci, G., and Mascolo, S. Experimental investigation of the google congestion control for real-time ows. In Proc. ACM SIGCOMM FhMN'13 (2013).

[13]. Fund, F., Wang, C., Liu, Y., Korakis, T., Zink, M., and Panwar, S. S. Performance of DASH and WebRTC video services for mobile users. In Proc. PV'13 (2013).

[14]. Hardie, T., Jennings, C., and Turner, S. Real-time communication in web-browsers. online, 2012. https://tools.ietf.org/wg/rtcweb/.

[15]. Homer, S., Lundin, H., Carlucci, G., Cicco, L. D., and Mascolo, S. A Google congestion control algorithm for real-time communication. IETF draft, 2015. https: //tools.ietf.org/html/draft-ietf-rmcat-gcc-01.

[16]. Jacobson, V. Congestion avoidance and control. In Proc. ACM SIGCOMM'88 (1988).

[17]. Johansson, I. Self-clocked rate adaptation for conversational video in LTE. In Proc. ACM SIGCOMM CSWS'14 (2014).

[18]. Mukherjee, D., Bankoski, J., Grange, A., Han, J., Koleszar, J., Wilkins, P., Xu, Y., and Bultje, R. The latest open-source video codec VP9-an overview and preliminary results. In IEEE PCS'13 (2013).

[19]. Nagy, M., Singh, V., Ott, J., and Eggert, L. Congestion control using FEC for conversational multimedia communication. In Proc. ACM MMSys'14 (2014).

[20]. Nam, H., Kim, K.-H., and Schulzrinne, H. QoE matters more than QoS: Why people stop watching cat videos. In Proc. IEEE INFOCOM'16 (2016).

[21]. Schulz-Zander, J., Mayer, C., Ciobotaru, B., Schmid, S., Feldmann, A., and Riggio, R. Programming the home and enterprise WiFi with OpenSDWN. In Proc. ACM SIGCOMM'15 (2015).

[22]. Shalunov, S., Hazel, G., Iyengar, J., and Kuehlewind, M. Low extra delay background transport (LEDBAT). IETF RFC 6817, 2012.

[23]. Singh, V., Lozano, A. A., and Ott, J. Performance analysis of receive-side real-time congestion control for WebRTC. In Proc. IEEE PV'13 (2013).

[24]. Wei, D. X., Jin, C., Low, S. H., and Hegde, S. FAST TCP: motivation, architecture, algorithms, performance. IEEE/ACM Trans. Netw. 14, 6 (2006), 1246{1259.

[25]. Winstein, K., Sivaraman, A., Balakrishnan, H., et al. Stochastic forecasts achieve high throughput and low delay over cellular networks. In Proc. USENIX NSDI'13 (2013).

[26]. Yiakoumis, Y., Katti, S., Huang, T.-Y., McKeown, N., Yap, K.-K., and Johari, R. Putting home users in charge of their network. In Proc. ACM UbiComp'12 (2012).

[27]. Zhu, X., and Pan, R. NADA: A uni_ed congestion control scheme for low-latency interactive video. In Proc. IEEE PV'13 (2013).