Performance Analysis of Google Congestion Control Algorithm for WebRTC

M.L. Guerrero Viveros

Performance Analysis of Google Congestion Control Algorithm for WebRTC

by

M.L. Guerrero Viveros

in partial fulfilment of the requirements for the degree of

Master of Science
in Electrical Engineering
Track Telecommunications and sensing systems

at the Delft University of Technology,
to be defended publicly on Thursday November 07, 2019 at 10:00 AM.

Student number:	4736605	
Project duration:	January 1, 2019 – November 07, 2019	
Thesis committee:	Prof. dr. ir. P. Van Mieghem,	TU Delft
	Dr. P. Pawelczak,	TU Delft
	Ir. R. Noldus,	Ericsson, supervisor

This thesis is confidential and cannot be made public until November 07, 2019.

Abstract

Web Real-Time communication (WebRTC) is a technology that enables web browsers to establish real-time communication services without the need of specific software or plug-ins. This technology is gaining popularity and is already supported by popular browsers such as Google Chrome, Firefox and Safari. The quality of real-time communication services depends highly on latency. For this reason, real-time flows have different requirements than conventional TCP flows which focus mainly on the transfer of bulk traffic. The IETF created the working group RMCAT (RTP Media Congestion Avoidance Techniques) to define requirements for real-time congestion control algorithms. One of the proposed algorithms is Google Congestion Control (GCC). This is the only real-time congestion control algorithm implemented in commercial browsers such as Google Chrome. Unfortunately, the performance of GCC in wireless networks has not been extensively evaluated. It is not clear yet what limitations a WebRTC communication might encounter in this type of networks, especially when it is competing with other type of flows.

This project addressed this issue by evaluating GCC in different technologies, namely in wired, WiFi and 4G networks. Controlled testbeds were used for the evaluation. The experiments followed the evaluation guidelines for real-time congestion control algorithms defined by the IETF. GCC proved to be a compliant RMCAT congestion control algorithm in networks with no contention. However, the results obtained in wireless access technologies revealed that GCC collapses when TCP flows are present in the channel. This issue is not attributed to GCC itself but to channel access methods of this type of networks. It is necessary to implement procedures to assign a different QoS to WebRTC flows in order to overcome this problem.

Preface

This thesis marks the finish line of my Master studies in Electrical Engineering at Delft University of Technology. The track of Telecommunications and Sensing Systems has been an exciting one. During my studies, I had the support from people around me that kept me going at a steady pace. Therefore, special thanks to Rogier Noldus for giving me the opportunity to do my thesis at Ericsson and his overall guidance. Thanks to René van der Mast for giving me the opportunity to join the VAS team as an Ericsson employee while finishing my Master thesis. I also want to thank Ericsson for letting me use the +31 network to carry out the tests that were necessary for the success of this thesis, and especially Jan Oude Vrielink for supporting me with that. Last but not least, I want to thank my family for their unconditional love and support.

M.L. Guerrero Viveros
Delft, November 2019

Contents

1 Introduction

The background, motivation and main objectives of this study are presented in sections 1.1, 1.2 and 1.3 respectively. A relevant literature review is presented in section 1.4. Finally, sections 1.5 and 1.6 present the contribution and outline of this work.

1.1. Background

In 2011, the IETF (RTCWEB Working Group) and W3C (WebRTC Working Group) started working on the standardization of an architecture and suite of protocols to enable real-time communication services between browsers. Nowadays, the Web Real-Time Communication (WebRTC) technology is available in popular browsers such as Google Chrome, Safari and Firefox. The open-source project WebRTC[1] of Google, is growing rapidly and already ranks fourth as the most popular technology for messaging (chat, video and voice) worldwide [1][2]. This implementation is used, among others, in Google Hangouts and Facebook messenger.

WebRTC is also seen by telecom providers as the key technology to enrich their real-time communication services specially since the Smartphone subscriptions, which now account for 90% of all mobile data traffic, are projected to reach 7.2 billion in 2024 [3]. Since smartphones are already fully capable multimedia devices, WebRTC applications on these devices can be used to massively deploy real-time communication services without any prior end-user intervention.

Quality of Experience (QoE) for real-time communication services depends highly on the latency, used bandwidth and experienced packet loss. This type of service has different requirements than conventional TCP's congestion control algorithms which mostly focus on the bulk transfer of non-critical data. These algorithms are also used for the transfer of media streams but in a non-interactive manner which again has different requirements than real-time services. The IETF working group "RTP Media Congestion Avoidance Techniques" (RMCAT) defines requirements for congestion control algorithms of real-time media streams. Some of these requirements are low end-to-end latency (no more than 100 ms), fairness to other flows, both real-time and TCP flows and ability to adapt the bit rate according to changing network conditions [4].

So far, three congestion control algorithms have been proposed within the RMCAT working group.

- Google Congestion Control (GCC) by Google
- Network Assisted Dynamic Adaptation (NADA) by Cisco
- Self-Clocked Rate Adaptation for Multimedia (SCReAM) by Ericsson

[1]The standard and Google's implementation are both named WebRTC

The IETF has not yet selected a specific congestion control algorithm for WebRTC. However, GCC is taking the lead in commercial applications, in fact, it is the only congestion control algorithm that has been adopted by popular web browsers. It is included in the official releases of Google Chrome and Firefox. Because of the above, this study will only focus on GCC.

1.2. Motivation

To the best of our knowledge, there is a lack of studies on the performance of GCC, specially in wireless and mobile networks. Most of these studies, which are found in the scientific community, come from the same authors of the algorithm and have been performed only in wired networks. A complete evaluation of GCC in different access technologies will indicate whether this algorithm is able to fulfill the requirements of congestion control for real-time media streams imposed by the IETF regardless of the type of network. It will also indicate which aspects of the algorithm need to be improved.

Ericsson, being one of the largest telecom vendors in the world, is currently looking into commercializing its WebRTC solutions. Therefore, they see this study as a reference to define requirements and constraints on their WebRTC products.

Another important aspect is the integration of WebRTC in mobile networks. We would like to study the architecture and procedures defined by the 3GPP and WebRTC solutions offered by Ericsson.

1.3. Objectives

The main objectives of this thesis are:

- Evaluate the performance of a WebRTC video call in wired, wireless and mobile networks.
- Identify architectures and procedures to integrate WebRTC in mobile networks proposed by the 3GPP and Ericsson.

1.4. Related Work

The performance of GCC was initially evaluated in [5], short after being proposed in the IETF RMCAT WG in 2013. The evaluation, which was performed in a controlled testbed, found that GCC traffic was being starved in the presence of TCP flows and was also unstable when competing against other GCC flows. After this study, the authors designed a new mathematical model for the algorithm that estimates the one-way queuing delay between two endpoints using a Kalman filter. This estimation is then compared against an adaptive threshold to adjust the sending bitrate [6]. Since then, the algorithm has evolved and the same authors, in [7], finally presented GCC as a fully compliant RMCAT algorithm for WebRTC. In this paper, a detailed explanation of the mathematical model is given, as well as, the reasoning behind the design of the adaptive threshold whose purpose is mainly to cope with cross traffic. The performance of GCC is also evaluated using a controlled wired testbed using the performance evaluation guidelines defined by RMCAT [8]. We will extend this performance evaluation by analyzing other network conditions in wired networks and including wireless and mobile testbeds.

With respect to other evaluations of GCC, we found in [9], a study on the influence that different queue types has on the quality of a WebRTC call. The authors proved that the droptail configuration is not enough to guarantee a stable WebRTC communication in the presence of TCP flows. Other queue types such as AQM and HTB presented better performance. We will only evaluate GCC with the droptail configuration since this queue management approach is widely adopted in access network devices. Another study that focuses on the video conference service is presented in [10]. The author evaluated GCC in a controlled wired network, but the tests do not follow the guidelines defined by RMCAT. Therefore, the results were not obtained from a common framework which is needed to compare the performance of GCC against other RMCAT algorithms. Our study will follow the guidelines defined by RMCAT.

Regarding the standardization status lead by RMCAT WG, a detailed analysis is found in [11]. The paper begins by presenting the general aspects of congestion control algorithms for real-time media. These aspects include general architecture and congestion detection approaches. Then, the main characteristics of the current proposed algorithms are described. As we mentioned previously, these algorithms are GCC, NADA and SCReAM. The authors of this paper qualitatively compare the results obtained by the algorithms' authors using the performance evaluation guidelines defined by RMCAT. With regards to GCC, it is concluded that this has the slowest convergence. On the other hand, it is the most stable once the link capacity is reached. We will corroborate these findings. A comparison between GCC and NADA in a simulated network is presented in [12]. This study revealed once again, the slow convergence of GCC.

The authors of [11] also mention that with regards to the standardization status of the proposed algorithms, GCC is behind SCReAM and NADA. GCC, at the time of writing, is still a draft version in IETF due to reported issues in its loss-based component. The loss-based component is mainly used by RMCAT algorithms when the buffers overflow. In this situation, it is not possible to estimate the queuing delay. We expect to see some issues with GCC in wireless and mobile networks in presence of TCP flows due to packet loss in the network.

With respect to the evaluation of GCC in wireless and mobile networks, we did not find studies presenting a detailed analysis on this topic. The paper found in [13], acknowledges the quality issues that WebRTC communications might have in WiFi due to packet loss and latency fluctuations on the channel. However, the paper does not present a performance analysis of GCC in WiFi. Instead, the authors propose to add mechanisms in the MAC layer of the WiFi link to improve the quality of the WebRTC communication. We will evaluate GCC in controlled WiFi and 4G networks.

From the current proposed RMCAT algorithms, SCReAM was the only one developed for wireless and mobile access technologies. This algorithm developed at Ericsson Research is optimized to achieve a good performance in wireless access such as LTE. A description of this algorithm is presented in [14]. This algorithm has been extensively tested by its developers over both a simulated internet bottleneck and an LTE system simulator [15] using the guidelines defined by RMCAT. We will use these results as a reference for comparison purposes.

1.5. Contribution

This project evaluates the performance of a WebRTC video call in wired, WiFi and 4G networks. This evaluation helps to identify areas of GCC that need to be improved in order to fulfill the requirements for real-time congestion control algorithms defined by the IETF RMCAT WG. The test cases follow the guidelines defined by RMCAT, therefore, this study can be used to compare the performance of GCC against other algorithms. The study also provides the state of the art of WebRTC in 3GPP and Ericsson.

1.6. Outline

The high-level approach of this project is presented below:

1. Perform a literature review on WebRTC and GCC.
2. Study the protocol stack of the WebRTC standard and get familiar with its API and procedures by creating a simple video call application.
3. Obtain deep understanding of the IMS architecture and procedures defined by the 3GPP,

as well as, specifications related to WebRTC.

4. Understand the architecture and functionality of current WebRTC solutions offered by Ericsson.
5. Design procedures and deploy tools to evaluate the performance of WebRTC following the guidelines defined by the IETF RMCAT WG.
6. Develop a prototype to establish a WebRTC video call between two laptops connected via an Ethernet cable.
7. Evaluate the performance of the WebRTC video call implemented in 6.
8. Develop a prototype to establish a WebRTC video call in a WiFi network. Deploy mechanisms to congest the radio channel.
9. Evaluate the performance of the WebRTC video call implemented in 8.
10. Establish a WebRTC video call in the 4G testbed of Ericsson. Deploy mechanisms to congest the radio channel.
11. Evaluate the performance of the WebRTC video call implemented in 10.
12. Identify limitations, derive conclusions and define future work based on the results.

The rest of this document is organized as follows. Chapter 2 provides a detailed description of WebRTC and GCC. Then, chapter 3 presents basic functionalities of IMS. The integration of WebRTC in IMS and WebRTC solutions of Ericsson are described in chapter 4. Chapter 5 presents the experimental setup for the evaluation of GCC. The analysis of the results is performed in chapter 6. Finally, chapter 7 provides the conclusions of this work.

2 WebRTC

This chapter provides an overview of the WebRTC standard, as well as, a description of the Google Congestion Control (GCC) algorithm.

2.1. WebRTC Standard

The standardization of WebRTC is still work in progress. At the time of writing, the protocol specification defined by the IETF working group "RTCWEB" is an Internet-Draft, whereas the Javascript API specified by the W3C working group "WebRTC" is a candidate recommendation [16][17]. These two specifications provide an environment to set up a real-time communication (audio, video and data) between browsers, mobile platforms and IoT devices. WebRTC non-browsers need only comply with the protocol specification (IETF).

Figure 2.1 presents the browser model for WebRTC defined by the IETF [16]. The Javascript application is embedded within a web page which is hosted in a web server and downloaded by the browser. This Javascript application uses the WebRTC Javascript API to access the Real Time Communication (RTC) functionalities of the browser. The on-the-wire protocols from the lower part transport the media path. These protocols must comply with the WebRTC protocol suite defined by the IETF. On the other hand, the protocol for the signaling path used for connection management and media negotiation is not specified. This path can use standard-based protocols such as SIP over WebSockets or a proprietary protocol. The on-the-wire protocols from the upper part transport the signaling communication between the servers which can be implemented with protocols such as SIP or XMPP.

In [16], six functionality groups are defined for the WebRTC protocol suite. We will use a similar approach to explain these protocols.

2.1.1. Data Transport, data framing and securing

The protocol stack of WebRTC is presented in figure 2.2. WebRTC implementations must be able to use UDP and both, IPv4 and IPv6.

Interactive Connectivity Establishment (ICE)

WebRTC applications must support the ICE protocol to deal with Network Address Translator (NAT) boxes and firewalls. The ICE protocol for UDP-based communication is specified in [18]. During the initial stage of this protocol, the ICE agents (peers) gather address candidates (IP addresses and ports). These candidates might be:

- Addresses from local interfaces.
- Addresses on the public side of NAT (reflexive address) collected with the Session Traversal Utilities for NAT (STUN) protocol.

Figure 2.1: Browser Model [16]

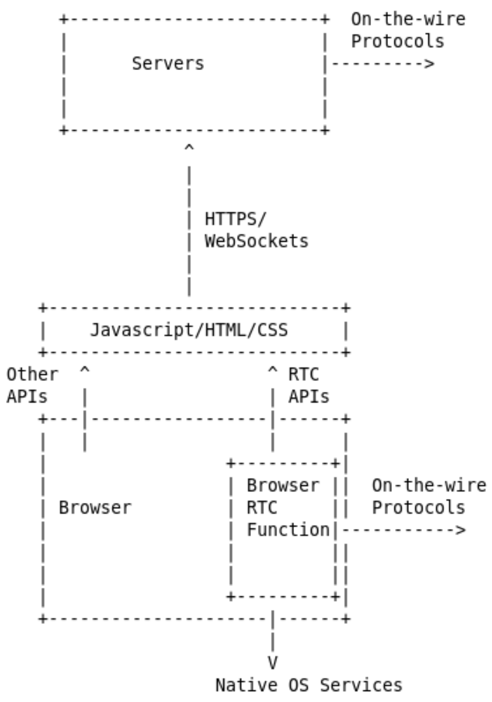
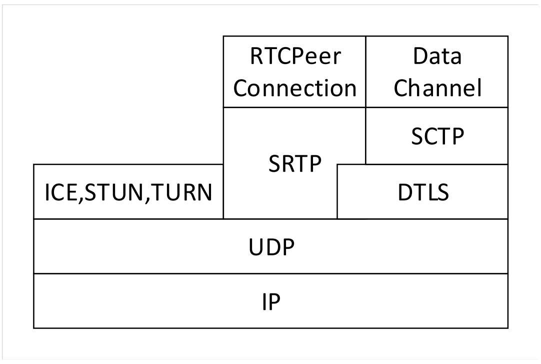


Figure 2.2: Basic protocol stack for WebRTC



- Addresses located in a Traversal Using Relay NAT (TURN) server (relayed address).

Then, each agent sorts its candidates by priority. Addresses from local interfaces have the highest priority, whereas relayed address have the lowest. The local candidate list is sent to the remote ICE agent (peer) through the signaling path. In WebRTC, this list is included in the Session Description Protocol (SDP) [19]. The local and remote candidates are paired up locally by each agent and then connectivity checks are performed for each candidate pair. Connectivity checks are STUN requests sent to the remote ICE agent. In case a STUN response is received, the candidate pair is marked as valid pair and added to a list of valid pairs. The ICE protocol selects one of the ICE agents as the controlling agent. This controlling agent selects one valid pair from the list of valid pairs and sends a new STUN request with an additional attribute to the remote ICE agent. This attribute indicates that the controlling agent wishes to use this valid pair to establish a peer session. The controlled ICE agent checks that this pair is also valid from his side and sends back a STUN response. After this, the valid pair becomes a selected pair that can be used to send and receive user data.

To keep the bindings alive at intermediate NATs during a session, ICE agents send periodic STUN requests to the STUN/TURN server. In order to accelerate the process of establishing a connection, WebRTC implementations must support Trickle ICE. This technique enables an ICE agent to start performing connectivity checks in parallel to candidate gathering [20].

Media Streams

The Real-Time Transport Protocol (RTP) is the main protocol for media transport in WebRTC [21]. WebRTC implementations must support both, RTP data transfer protocol and the RTP control protocol (RTCP). The Extended Secure RTP Profile for RTCP-Based Feedback (RTP/SAVPF) is the RTP profile that must be implemented for WebRTC [22]. This profile is a combination of basic RTP/AVC profile, the RTP profile for RTCP-Based feedback (RTP/AVPF) and the secure RTP profile (RTP/SAVP). The secure RTP profile (SRTP) is used to provide a limited form of source authentication, integrity and confidentiality to the media streams [23]. The keys used in SRTP are exchanged using DTLS-SRTP [24].

For a multimedia session, WebRTC implementations are required to send all RTP streams in a single RTP session using a single transport layer flow. Only one DTLS-SRTP handshake is needed for all RTP streams (one per session). Additionally, the multiplexing of RTP and RTCP packets on a single transport layer flow is required. For compatibility with legacy systems, WebRTC implementations can optionally support the transport of separate RTP and RTCP packets. To facilitate NAT traversal, Symmetric RTP is required to be supported.

Although, the protocol specification acknowledges the importance of implementing a real-time congestion control algorithm to deal with variable conditions on networks and cross traffic (i.e. other real-time traffic and TCP/SCTP traffic), at the time of writing, the specification does not define a congestion control algorithm for WebRTC media streams. However, three algorithms have been proposed within the RMCAT working group. These are:

- Google Congestion Control (GCC) by Google [25].
- Network Assisted Dynamic Adaptation (NADA) by Cisco [26].
- Self-Clocked Rate Adaptation for Multimedia (SCReAM) by Ericsson [15].

GCC will be explained in detail in a later section.

Data Channels

In WebRTC, non-media communication is transported using the Stream Control Transmission Protocol (SCTP) over the Datagram Transport Layer Security (DTLS) protocol [27][28]. WebRTC implementations must support DTLS 1.0 and DTLS 1.2, and the partial reliability extension of SCTP [29]. DTLS provides source authentication, integrity and confidentiality to the non-media communication. The multiplexing of DTLS and SRTP over the same transport layer flow must be supported. This is to facilitate NAT traversal.

In a WebRTC communication, a single SCTP association is created by the WebRTC peers (by exchanging SDP in the signaling path). The DTLS connection used for exchange of the SRTP keys is shared with this SCTP association. Multiple SCTP streams which are unidirectional logical channels can be created within a SCTP association. These SCTP streams share the same congestion window (TCP-friendly congestion control applies to the SCTP association). SCTP defines procedures to gracefully or abruptly close a connection. The latter could be performed when SCTP detects multiple retransmissions on the connection. A data channel is a pair of incoming and outgoing SCTP streams with the same SCTP stream identifier. Some properties of this channel are:

- Reliable or partially-reliable message transmission.
- In-order or out-order message delivery.
- Priority. This must be interpreted as weighted-fair-queuing scheduling priority.

WebRTC implementations must provide a way to configure these properties in a WebRTC application. The creation of a data channel in a WebRTC communication can be performed using an in-band or out-band negotiation. In-band negotiation is performed via the SCTP association. A simple protocol for in-band negotiation is specified by the IETF in [30]. Out-band negotiation is not specified in the protocol specification. According to [31], out-band negotiation is very useful in applications with many participating peers. The configuration can be simultaneously distributed to all parties in the signaling path.

Finally, it is important to note that WebRTC implementations must be able to demultiplex packets arriving on the same UDP port. These arriving packets could be SRTP, RTCP, DTLS or STUN packets. Finally, the protocol specification also defines a model for the prioritization of media and data streams [32].

2.1.2. Data formats

The WebRTC protocol specification does not restrict the usage of any audio or video codecs. However, there is a minimum set of codecs and profiles that must be supported by WebRTC implementations. Some audio codecs that must be supported are OPUS, PCMA and PCMU. For video codecs, VP8 and H.264 must be supported. The complete list of codecs and profiles that shall be supported is found in [33][34].

2.1.3. Connection Management, presentation and control

WebRTC media negotiation uses the same Session Description Protocol (SDP) offer/answer model that is used by the Session Initiation Protocol (SIP). WebRTC browsers must implement the Javascript Session Establishment Protocol (JSEP) which controls the signaling plane of a multimedia session via the W3C Javascript API [35]. JSEP exposes simple method calls on the RTCPeerConnection interface to handle the WebRTC media negotiation using SDP. WebRTC non-browsers must only implement functions described in JSEP related to the network layer such as the ability to negotiate media multiplexing with SDP, multiplexing of RTP and RTCP, and ICE Trickle.

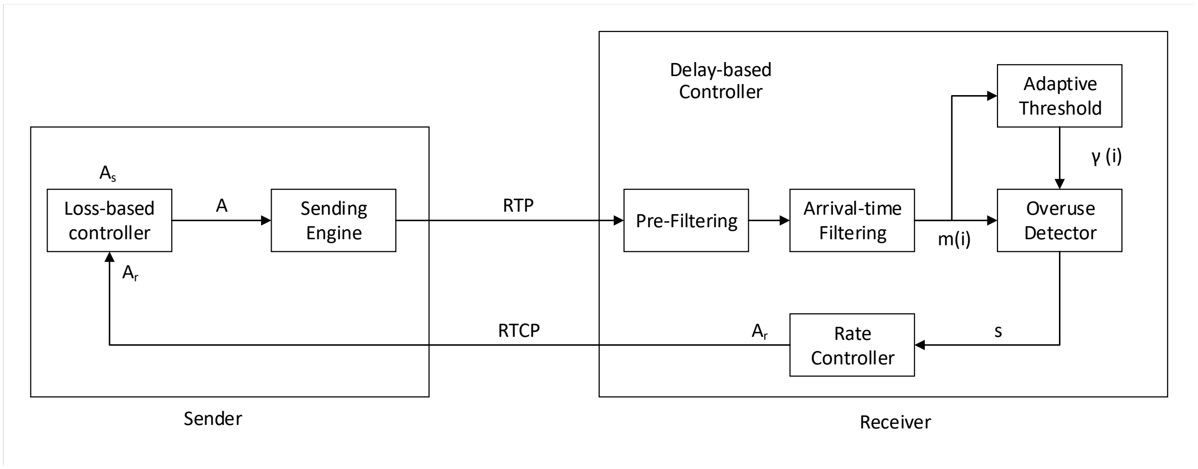
The W3C specifies the Javascript API used to set up and maintain WebRTC peer-to-peer communications via the browser [17]. This API abstracts all the inner workings of the protocols mentioned above behind a few interfaces that can be easily used by developers to create WebRTC applications. The RTCPeerConnection interface encapsulates the set up, negotiation, maintenance, state and termination of connections in a single interface. The W3C also specifies an API to access local devices [36]. WebRTC non-browsers can offer to developers APIs in other languages.

2.2. Google Congestion Control (GCC)

The Google Congestion Control (GCC) algorithm is one of the three protocols that have been proposed within the RMCAT working group. At the time of writing, GCC is an Internet-Draft specified in [25]. The main target of GCC is to adjust the sender bit rate when congestion or packet loss is detected. The algorithm tries to maintain a low queuing delay and shares the bandwidth with other real-time and loss-based streams as fair as possible.

GCC controls congestion using two controllers, a delay-based controller and a loss-based controller. The architecture of GCC is presented in figure 2.3. In this architecture, the delay-based controller is located at the receive-side. This controller monitors the incoming packets and calculates the available bit rate ($\hat{A}_r$) and packet loss ratio. This information is then sent back to the sender via RTCP receiver reports [37]. At the send-side, the loss-based controller uses the packet loss ratio to calculate its own available bit rate ($\hat{A}_s$). Finally, these two bit rates are compared and the minimum value is set as the target bit rate (A). This bit rate is used by the sending engine to send further RTP packets. It is also possible to have both controllers at the send-side. In this scenario, the receiver sends back to the sender (delay-based controller) the timestamp, size and sequence number of incoming packets via RTCP receiver reports.

Figure 2.3: GCC Architecture



2.2.1. Sending Engine

The final output of the controllers is a target bit rate which is enforced by the sending engine. In every burst_time interval, which is recommended to be 5 ms, the sending engine transmits a group of packets whose size is equal to the product of the target bit rate and burst_time.

2.2.2. Delay-based controller

GCC aims to fully utilize the available capacity of the link while keeping a low queuing delay. The queue length could be estimated from the one-way delay or the RTT (Round-Trip Time), however, these measurements are affected by different factors such as reverse path congestion and latecomer phenomenon. To avoid these issues, GCC uses the inter-group delay variation $d(i)$, which is defined as:

$$d(i) = (t_i - T_i) - (t_{i-1} - T_{i-1}) = (t_i - t_{i-1}) - (T_i - T_{i-1}) \tag{2.1}$$

In this equation, T_i corresponds to the time stamped in the last packet of the current packet group being sent. The inter-departure time, $T_i - T_{i-1}$, is the difference in departure time of two groups of packets. On the other hand, t_i is the arrival time stamped in the last packet of the received packet group. The inter-arrival time, $t_i - t_{i-1}$, is the difference in arrival time of two groups of packets.

In general terms, the delay-based controller increases the bit rate until over-utilization of the link is detected, then it decreases the bit rate to empty the queue. When under-utilization is detected, the bit rate is again increased. This controller is composed of four components, namely a pre-filtering function, an arrival-time filter, an over-use detector and a rate control. We will describe these components in the following subsections.

Pre-filtering

The pre-filtering is used when a burst of packets is sent due to channel outages. With this function, it is possible to identify which packets belong to a certain group of packets. One of the following two rules must hold for packets to be merged in one group.

- All packets which are sent within a burst_time interval constitute a group.
- A packet with an inter-arrival time less than burst_time and inter-group delay variation less than 0 is merged in the current group of packets.

Arrival-time filter

The arrival-time filter calculates the inter-group delay variation as stated in equation 2.1. This delay variation is a function of the transmission time variation, one-way queuing delay variation $m(i)$ and noise (network jitter and other not captured delay effects). A Kalman filter is then used to estimate the one-queuing delay variation $m(i)$. This filter eliminates both the noise and the contribution of the transmission time variation [7].

Over-use detector

The one-queuing delay variation $m(i)$, estimated by the arrival-time filter, is compared with a threshold $\gamma(i)$. The recommended initial value for this threshold is 12.5 ms. When $m(i)$ remains above this threshold for $overuse\_time\_th$ milliseconds (recommended value 10 ms), the detector signals over-use to the rate control subsystem. Similarly, under-use is signaled when $m(i)$ remains below the threshold $-\gamma(i)$ for $overuse\_time\_th$ milliseconds. The detector remains in normal state when $m(i)$ is between the interval $[-\gamma(i),\ \gamma(i)]$.

In order to avoid starvation due to the presence of a concurrent TCP flow, it is necessary to dynamically adjust the value of the threshold $\gamma(i)$. The general idea is to increase this threshold when the one-queuing delay variation increases (possibly due to the presence of a TCP flow). This leads to a reduction of the number of over-use signals triggered to the rate control subsystem, which in turn reduces the starvation of the real-time media flow. The equation to dynamically adjust this threshold is presented below.

$$\gamma(i) = \gamma(i-1) + (t(i) - t(i-1)) * K(i) * (|m(i)| - \gamma(i-1)) \tag{2.2}$$

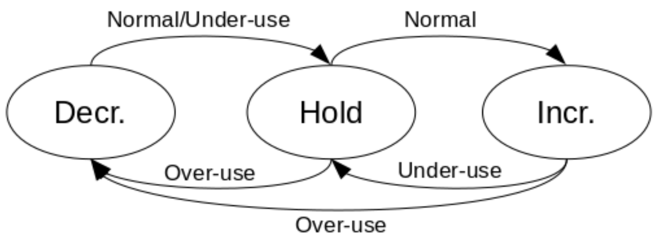
$$K(i) = \begin{cases} K_d = 0.00018 & |m(i)| < \gamma(i-1) \\ K_u = 0.01 & Otherwise \end{cases} \tag{2.3}$$

The threshold $\gamma(i)$ is increased when $m(i)$ is outside the interval $[-\gamma(i-1),\ \gamma(i-1)]$ and decreased otherwise. The recommended values for $K(i)$ are presented in equation 2.3. The value of $\gamma(i)$ is not updated if $|m(i)| - \gamma(i-1) > 15$. A complete description of the adaptive threshold design is presented in [7].

Rate control system

The finite state machine of the rate control system is presented in figure 2.4. The control signal "s" is the output of the over-use detector. The system starts in the increase state in which the available bandwidth $\hat{A}_r$ is increased either additively or multiplicatively. When over-use is detected, the state changes to decrease. In this state, $\hat{A}_r$ is decremented until the signal under-use is emitted. At this point, the system changes to the hold state in which $\hat{A}_r$ is maintained constant to drain the built-up queues along the path. When $m(i)$ is close to zero, the system goes again to the increase state.

Figure 2.4: FSM Rate Control System



The available bandwidth $\hat{A}_r$ is increased multiplicatively if the incoming bitrate $\hat{R}(i)$ is three standard deviations below the average bitrate measured when the system was last in the decrease state. In this case, we say that the system is far from convergence. The multiplicative mode increases $\hat{A}_r$ by at most 8% per second. On the other hand, the additive mode increases $\hat{A}_r$ by at most half a packet per response_time interval. The response_time interval is the round-trip time plus 100 ms. If the incoming bit rate is three standard deviations above the previous average bitrate, the rate control switches to multiplicative mode. This is an indication that the available capacity has changed. In both modes, $\hat{A}_r$ is upper bounded to $1.5 * \hat{R}(i)$ in case the sender is not able to produce the target bitrate the controllers are asking for.

When over-use is detected, $\hat{A}_r$ is decreased to a factor times the currently incoming bitrate $\hat{R}(i)$, $\hat{A}_r(i) = \mathrm{B} * \hat{R}(i)$. This factor B is recommended to be 0.85. In the hold state $\hat{A}_r$ is kept constant.

2.2.3. Loss-based controller

The available bandwidth $\hat{A}_r$ estimated by the delay-based controller is only valid when the queues along the path are sufficiently large. When the queues are short, packet loss is the only way to detect over-use. The loss-based controller estimates its own available bandwidth $\hat{A}_s$ based on packet loss thresholds. The target bit rate used by the sending engine is set to the minimum of $\hat{A}_r$ and $\hat{A}_s$. The available bandwidth $\hat{A}_s$ is calculated according to equation 2.4, where p is equal to the loss ratio.

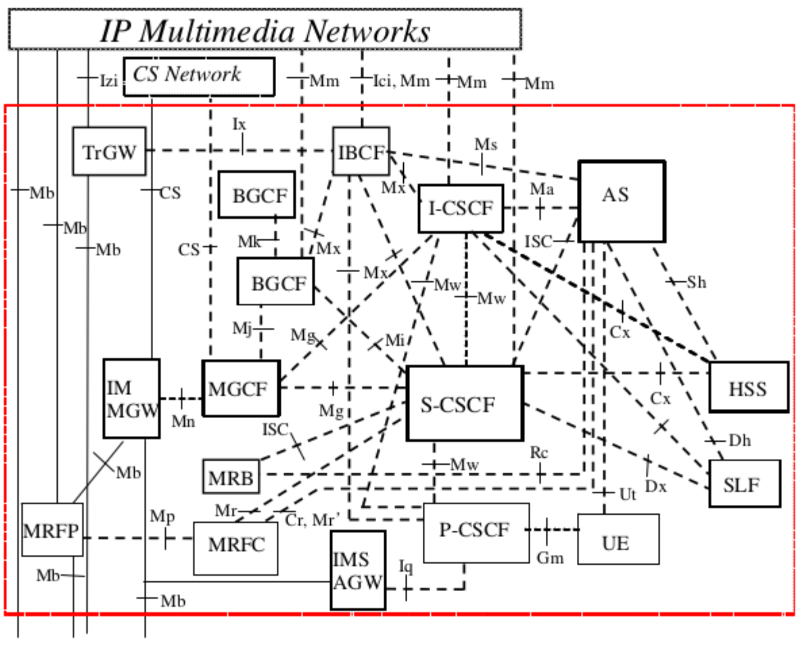
$$\hat{A}_s(i) = \begin{cases} 1.05 * \hat{A}_s(i-1) & packetloss < 2\% \\ \hat{A}_s(i-1) & 2\% < packetloss < 10\% \\ \hat{A}_s(i-1) * (1 - 0.5 * p) & packetloss > 10\% \end{cases} \tag{2.4}$$

3
IP Multimedia Subsystem (IMS)

The aim of this chapter is to provide a high level description of the IP Multimedia Subsystem (IMS) in order to understand its basic architecture and main procedures. IMS is used by operators to offer multimedia services to their subscribers. Therefore, WebRTC is seen as a technology that will be integrated in the IMS network to improve existing real-time communication services such as Voice over LTE (VoLTE). IMS will also enable the creation of new communication services based on WebRTC.

IMS is access agnostic, meaning that any wireless or wireline access technology can be used to access the IMS network to provide voice, video, messaging and data services to subscribers. The Session Initiation Protocol (SIP) was selected as the signaling protocol within IMS and RTP as the protocol to transport voice and video (user plane). The reference architecture of IMS is presented in figure 3.1 [38]. A description of nodes and reference points is found in [39]. An explanation of IMS architecture and protocols from an application developer's perspective is found in [40]. Readers are refered to this document for a detailed explanation of SIP, SDP and SDP offer/answer exchange.

Figure 3.1: Reference architecture of IMS [38]



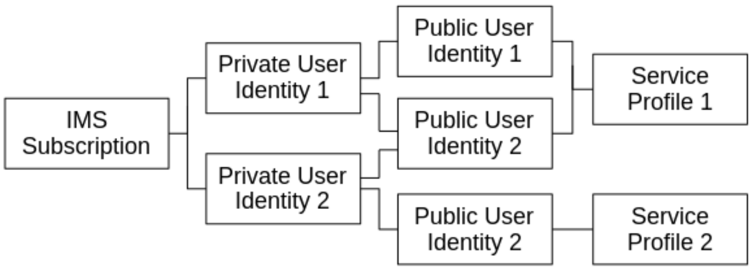
3.1. Identity Names

The two main identities associated with a user of IP multimedia services are the Private User Identity (IMPI) and the Public User Identity (IMPU). Private User Identities are permanent identities associated to a user's subscription which are used for registration, authorization, administration and accounting purposes. This identity has the form username@realm and is provided by the home network operator. The IP Service Identity Module (ISIM) shall store one IMPI. When ISIM is not present in the UICC (Universal Integrated Circuit Card), the IMPI can be generated from the International Mobile Subscriber Identity (IMSI) [41]. It is important to mention that IMPI is not used for routing of SIP messages.

Public User Identities are identities used by users for requesting communication to other users. Theses identities can take the form of a SIP URI (e.g. "sip:username@domain") or a Tel URI (e.g "tel:+311234567"). A user shall have one or more IMPUs where one must be a SIP URI. In case a telephone number is used as an identity, a Tel URI or a SIP URI with a "user=phone" URI parameter must be used. The telephone number can be a global number in E.164 format or a local number with a "phone-context" parameter to identify the scope of its validity [41]. The ISIM must store at least one IMPU. In case the ISIM is not present, a temporary IMPU can be generated from the IMSI. These user identities are stored in the Home Subscriber Sever (HSS) either as distinct IMPU or as wildcarded IMPU (wIMPU). Distinct Public User Identities are used for routing and are explicitly provisioned in the HSS, whereas a wIMPU represents a group of IMPUs that share the same service profile and implicit registration set.

An IMS subscription may have multiple IMPIs and IMPUs. One or more IMPUs may be shared across these IMPIs. The relationship of user identities in an IMS subscription is presented in figure 3.2.

Figure 3.2: Relation of user identities in an IMS subscription



Additionally, services hosted in Application Servers (ASs) are identified by using Public Service Identities (PSIs). PSIs can take the form of SIP URIs or Tel URIs. These identities are stored in the HSS as distinct PSIs or wildcarded PSIs. The former are used for routing, whereas a wildcarded PSI represents a collection of PSIs. Wildcarded PSIs shall take the form of Extended Regular Expressions whose delimiter is the exclamation mark character (!). For instance, the wildcarded PSI "sip:webrtc!.*!@example.com" will match "sip:webrtc1@example.com" or "sip:webrtc2@example.com".

3.2. Entities Roles

The following subsections describe the roles of the Call Session Control Function (CSCF), as well as, other basic entities of IMS.

3.2.1. Proxy-CSCF

The Proxy-CSCF (P-CSCF) is the entry point for the User Equipment (UE) to the IMS. It behaves like a SIP proxy. Its main functions are:

- Forward SIP register requests from the User Equipment (UE) to the I-CSCF based on the home domain name provided by the UE.
- Forward SIP messages from the UE to the S-CSCF received during the registration process.
- Forward SIP messages to the UE.
- Maintain a security association between the UE and itself.
- Filter and adapt SIP messages sent by the UE to comply with operator policies.
- Interact with the Policy and Charging Rules Function (PCRF) for QoS management.
- Interact with the IMS-AGW (IMS Access Gateway) for media plane control.

The UE can obtain the IP address of the P-CSCF in multiple ways. One way to obtain this address is during the establishment of connectivity towards the access network. If the address is not provided during this procedure, the P-CSCF can be discovered by using DHCP/DNS queries. Finally, the IP address can also be previously provisioned in the ISIM, via a 3GPP IMS Management Object (MO) or in the phone application.

3.2.2. Interrogating-CSCF

The Interrogating-CSCF (I-CSCF) is the entry point for messages destined for a user of that network operator (in case border control is applied, IBCF becomes the entry point). Its main functions are:

- Assign a S-CSCF during the registration procedure.
- Route a SIP request towards the S-CSCF assigned to that user. The S-CSCF is obtained from the HSS. In case the domain does not belong to the home network, I-CSCF can perform transit routing procedures if configured.
- Translate the E.164 number contained in a SIP-URI with ”user=phone” parameter into a Tel URI format before performing the HSS location query. If the HSS indicates that the user does not exist, I-CSCF can perform transit routing procedures if configured.

3.2.3. Serving-CSCF

The Serving-CSCF (S-CSCF) is the core entity of IMS. Some of its main functions are presented below.

- Behave as a registrar for subscribers. Notify the HSS that a user has registered.
- Notify subscribers about registration changes.
- Determine the next hop for a SIP request or response.
- Analize filter criteria to forward SIP requests or responses to ASs.
- Provide other IMS nodes with service event related information (e.g. billing notification).

3.2.4. Breakout Gateway Control Function (BGCF)

When the S-CSCF or I-CSCF (if transit routing procedures are configured) determines that a SIP message cannot be routed using DNS or ENUM/DNS (translation of E.164 number to SIP URI), the BGCF is in charge of finding the next hop for this message. The routing is based on protocol information, administrative configuration and/or database access. If BGCF determines that the next hop is a local PSTN/CS network, the message is routed towards a Media Gateway Control Function (MGCF). If the next hop is an external network, the message is sent out to that network. In case the routing determination results in a break out in another IMS network, the message is sent to the I-CSCF (or IBCF) of that network. Lastly, a specific IP address can be configured as the next hop.

3.2.5. Interconnect Border Control Function (IBCF)

When configured by an operator, IBCF is in charge of performing border control functions between two IMS networks. IBCF becomes the entry point for a network (instead of I-CSCF). Some functions of this entity are:

- Establish communication between two IMS networks using different media codecs.
- Provide network configuration hiding to restrict sensitive information from being passed.
- Interconnect IMS networks with different signaling protocols (e.g. SIP and H.323) by interacting with the Transition Gateway (TrGW).
- Perform transit routing procedures if configured.
- Interact with the TrGW for media plane control.

3.3. Registration

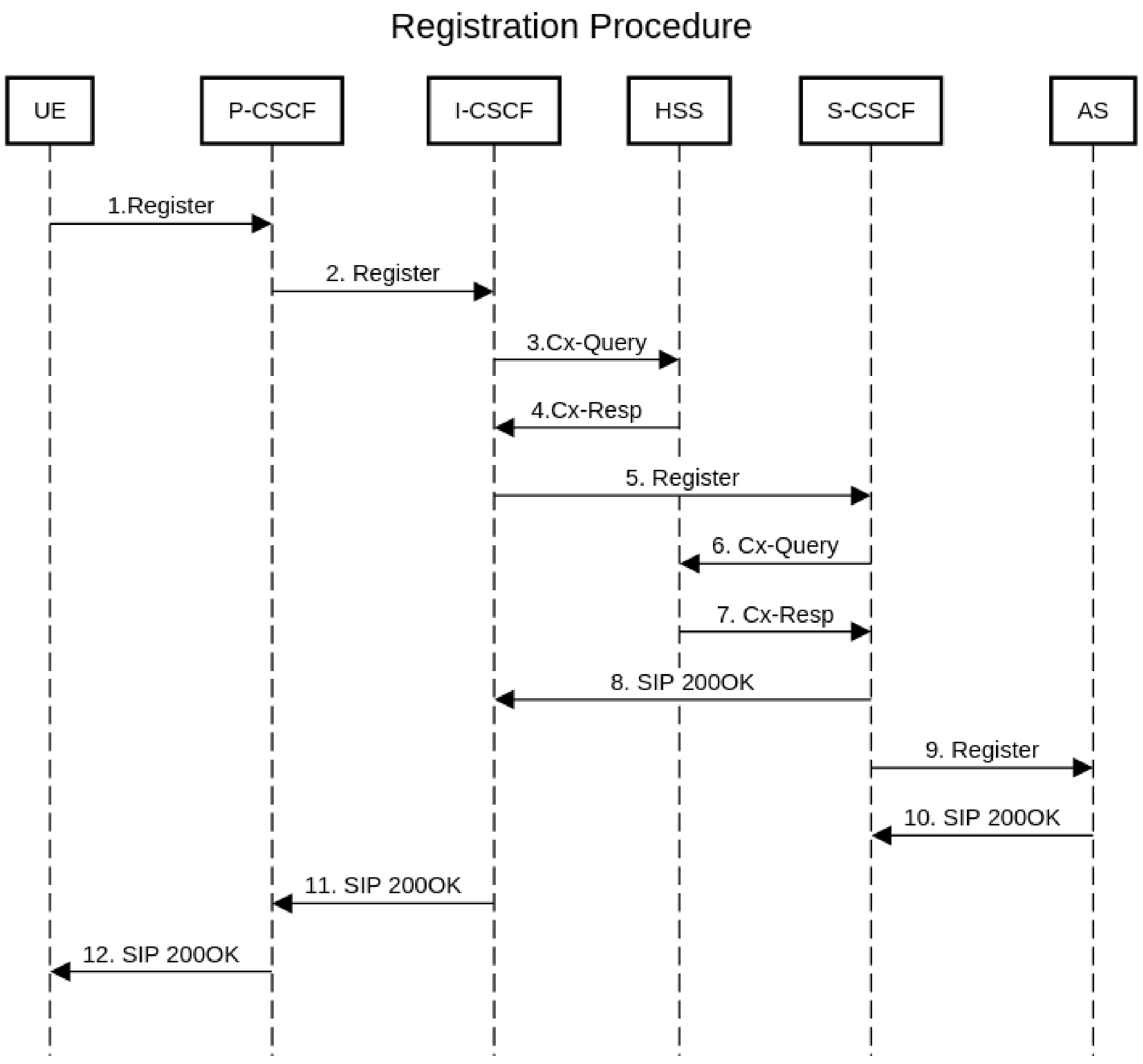
Registration to the IMS network is a requirement for a UE to be able to establish originating and terminating IMS sessions. Although, this procedure also allows the network to authorize and authenticate users, these features will not be included in this section. Please refer to 3GPP TS 33.203 [42] for information on IMS security.

Before describing the registration procedure, it is important to define what implicit registration set means. An implicit registration set is a group of IMPUs that have different service profiles or with some IMPUs pointing to the same service profile. When an IMPU that belongs to an implicit registration set is registered (de-registered), all other IMPUs belonging to that implicit registration set are also registered (de-registered).

Figure 3.3 presents the registration information flow. IMS security features are not considered.

1. After obtaining IP connectivity, the UE sends the SIP REGISTER request message to the P-CSCF.
2. The P-CSCF checks the validity of the request (e.g. checks that only one contact is present). The P-CSCF examines the home domain name sent by the UE to obtain the entry point to the home network (I-CSCF). A DNS query is performed to determine the IP address of this home domain name. Before forwarding this message to the I-CSCF, a Path header is added so that the S-CSCF can route terminating requests back to this P-CSCF. A P-Visited-Network-ID, used to identify the visited network, is also added.
3. The I-CSCF queries the HSS. This message includes the IMPU, IMPI and P-Visited-Network-ID.
4. The HSS checks, among others, that the IMPU belongs to the network home domain and is not a PSI. A response is sent back to the I-CSCF with the name of the S-CSCF if the user is already registered or S-CSCF capabilities.
5. The I-CSCF selects, if needed, a S-CSCF using the S-CSCF capabilities. A Route header with the URI of the selected or provided S-CSCF is added. A DNS query is performed to resolve this name.
6. The S-CSCF receives the SIP REGISTER and notifies the HSS that is now serving this user. A query is sent to the HSS including the IMPU, IMPI and name of the S-CSCF.
7. The HSS stores the name of the S-CSCF and sends back the IMPU being registered and (if applicable) all other IMPUs in an implicit registration set. Also, the service profiles of the IMPUs are sent to the S-CSCF including Service Triggers and Initial Filter Criteria (iFC).

Figure 3.3: Registration Procedure



8. The S-CSCF stores the user data of all received IMPUs and the IMPI. A SIP 200(OK) is returned to the I-CSCF. This messages includes, among others, a P-Associated-URI header with a list of all IMPUs that have been registered and a Service-Route header with the domain name of the S-CSCF.

9. A 3rd party registration is sent to the AS based on filter criteria.

10. The AS processes the SIP REGISTER and sends back a SIP 200(OK).

11. The I-CSCF proxies the 200(OK) response.

12. The IMPUs found in the P-Associated-URI are stored in the P-CSCF. The first entry is the default IMPU. The P-CSCF can subscribe to notifications of the status of the IMS signaling connectivity from the PCRF via the Rx interface. A 200(OK) is sent to the UE.

3.4. End-to-End Session Flow Procedures

End-to-end session flows have different procedures for each type of origination and termination sequences (e.g. MO (Mobile Origination), MT (Mobile Termination), PSTN, MT CS

(Circuit Switched) domain, AS (Application Server)). These procedures also vary if the session origination and termination are served by the same or different network operators and if the session requires or not, preconditions for the establishment of QoS-Assured sessions [38]. For the purpose of this project, Mobile Origination-home (MO,home) and Termination-home (MT,home) sessions with preconditions are explained in the following subsections. It is assumed that both users are subscribed and located in the same network. A complete description of other procedures can be found in [38][43].

3.4.1. Mobile Origination, home

It is assumed that a user located in his/her home service area wants to establish a session with preconditions towards a user who also supports preconditions. The P-CSCF remembers the S-CSCF allocated to this user during registration and the S-CSCF, likewise, knows the address of the P-CSCF serving this user. Figures 3.4 and 3.5 present the mobile origination procedure.

1 The UE sends a SIP Invite request to the P-CSCF assigned during registration. The Request-URI header is the SIP or TEL URI of the target user. The Contact header is the IP address of the UE. A Support header is added to indicate that precondition negotiation is supported. The SDP offer includes attributes to indicate that local resources are not available.

2 The P-CSCF receives the request and checks that this has been sent by a registered user. A P-Asserted-Identity header is added to the message. The value is one of the IMPUs of the calling user. The SIP request is forwarded to the S-CSCF stored at registration.

3 The S-CSCF checks that the user is registered. If the P-Asserted-Identity header is part of an Implicit Registration Set (IRS) then a second P-Asserted-Identity is added. If the first header is a TEL URI then a SIP URI is added. If the first header is a SIP URI and if there is a valid TEL URI, then a TEL URI is added. A P-Served-User is also included in the message. The value is the IMPU present in the P-Asserted-Identity. This header also contains the session case and user's registration case. For this scenario, ”originating” and ”registered” are the corresponding values. Then, the S-CSCF analyzes the initial Filter Criteria (iFC) to check if there is a match. If this is the case, the request is sent to the corresponding AS. A Route header is added pointing back to the S-CSCF.

4 The AS processes the request and sends it back to the S-CSCF.

5 In case the Request-URI has a telephone number, number normalization is used to assure that the number is in international E.164 format. A DNS NAPTR query is then used to convert the telephone number to a SIP URI. The S-CSCF sends the request to the next hop, which due to the assumptions made in this example, is an I-CSCF located in the same network. In other scenarios, this message might be sent to a BGCF or IBCF located in the same network or directly to an I-CSCF of another IMS network.

6 The next hop (I-CSCF) sends a SIP 183 Session Progress response to the S-CSCF. This message indicates that precondition negotiation must be used. The SDP answer includes attributes to indicate that remote resources are not available.

7-9 The SIP 183 Session Progress response is forwarded to the P-CSCF.

10 The P-CSCF sends a RX Diameter AAR request to the PCRF to request resources in the access network.

11 The PCRF checks if the request fulfills local policies and sends back a RX Diameter AAA answer.

12 The P-CSCF forwards the SIP 183 response to the UE.

13-17 The UE sends a SIP PRACK to acknowledge the SIP 183 response.

Figure 3.4: Originating Procedure (part 1)

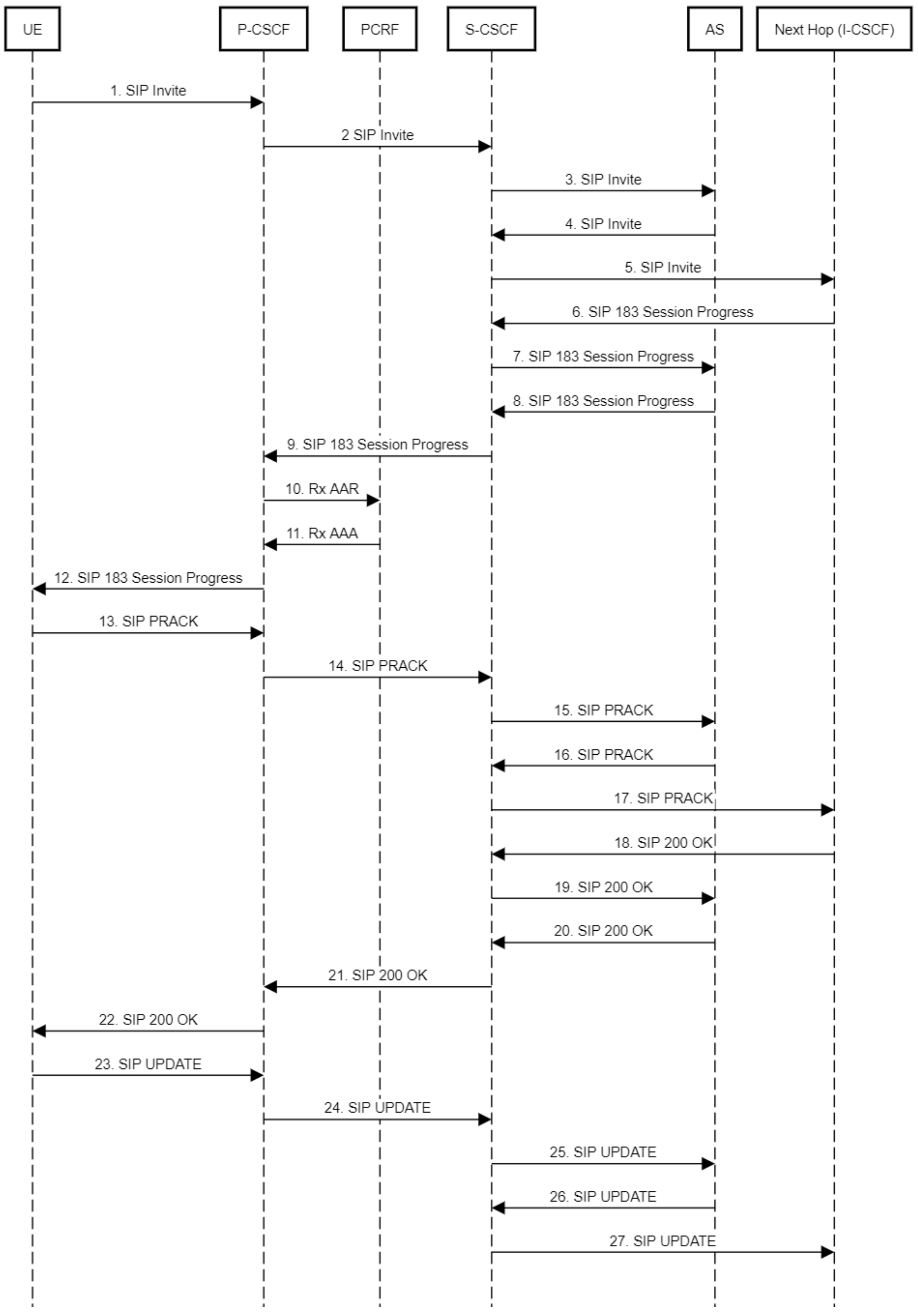
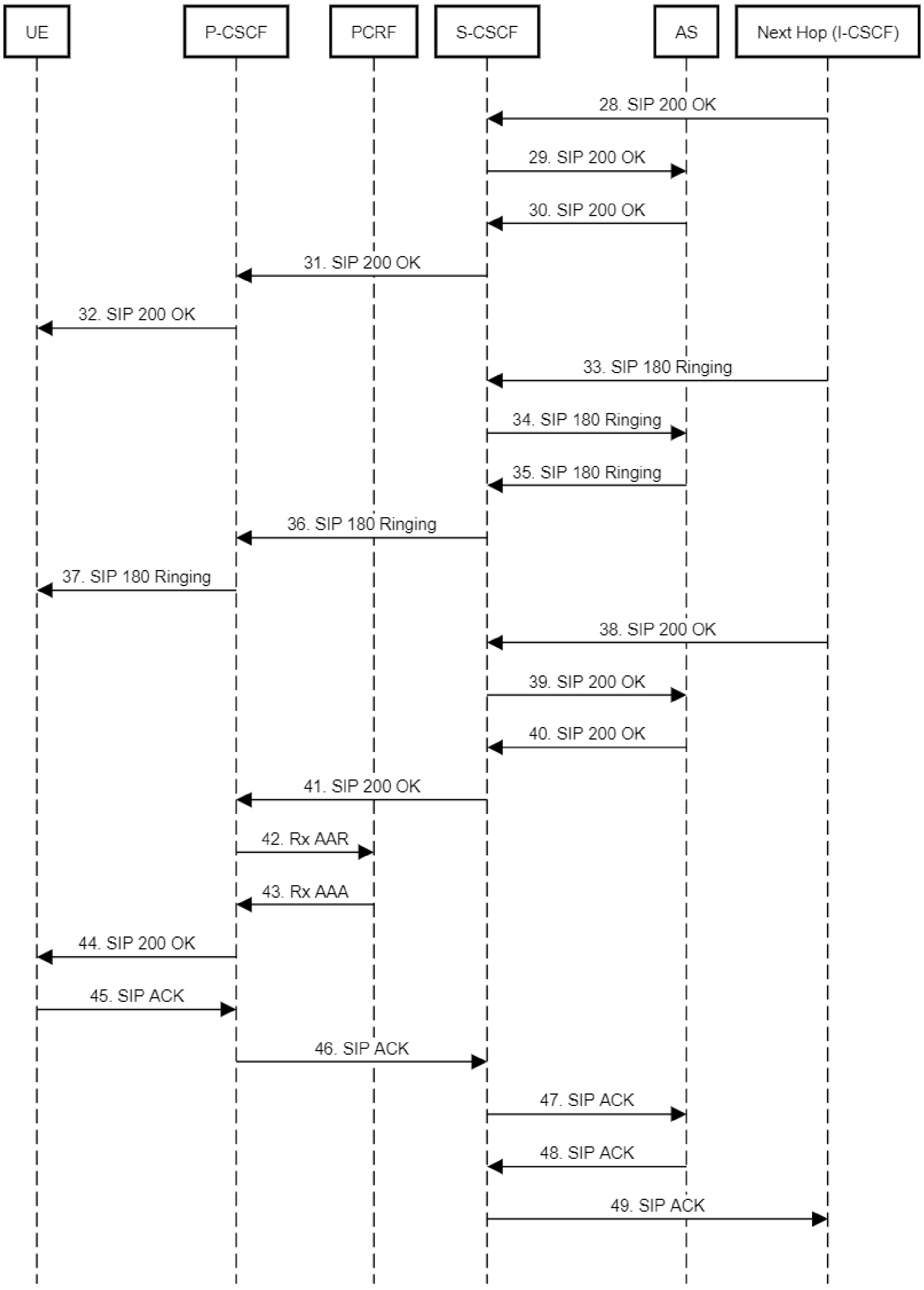


Figure 3.5: Originating Procedure (part 2)



18-22 The next hop sends a SIP 200 OK to acknowledge the SIP PRACK.

23-27 The UE sends a SIP UPDATE request with an SDP offer indicating that local resources are available.

28-32 The next hop sends a SIP 200 OK to acknowledge the SIP UPDATE request. The SDP offer, which is included in this message, indicates that the remote resources are available.

33-37 The next hop sends a SIP 180 Ringing response.

38-41 The next hop sends a SIP 200 OK to the P-CSCF.

42-43 The P-CSCF sends media information to the PCRF to allow the connection in both directions. It also indicates that this information is final.

44 The P-CSCF sends the SIP 200 OK to the UE.

45-49 The UE sends a SIP ACK

3.4.2. Mobile Termination, home

It is assumed that a user located in his/her home service area receives a request to initiate a session with preconditions. The P-CSCF remembers the S-CSCF allocated to this user during registration and the S-CSCF, likewise, knows the address of the P-CSCF serving this user. Figures 3.6 and 3.7 present the mobile termination procedure.

1 The I-CSCF receives the SIP Invite request from the previous hop. For this case, the previous hop is the S-CSCF.

2 If the Request-URI is a SIP URI that includes a ”user=phone” parameter, the I-CSCF converts this SIP URI to a TEL URI before querying the HSS. The S-CSCF sends a Location Info Request (LIR) message to the HSS. This message includes the IMPU present in the Request-URI.

3 Since for this example the called user is known and registered, the HSS sends back a Location Info Answer (LIA) message with the name of the S-CSCF serving the called user.

4 The I-CSCF forwards the SIP Request to the S-CSCF.

5 The S-CSCF analyzes the initial Filter Criteria (iFC) to check if there is a match. If this is the case, the request is sent to the corresponding AS. A Route header is added pointing back to the S-CSCF.

6 The AS processes the request and sends it back to the S-CSCF.

7 The S-CSCF sends the request to the P-CSCF stored at registration.

8 The P-CSCF checks that the user is registered and stores the value of the P-Called-Party-ID header. The request is then sent to the UE.

9 The UE sends a SIP 183 Session Progress response to the P-CSCF. This message indicates that precondition negotiation must be used. The SDP answer includes attributes to indicate that local resources are not available.

10 The P-CSCF sends a RX Diameter AAR request to the PCRF to request resources in the access network.

11 The PCRF checks if the request fulfills local policies and sends back a RX Diameter AAA answer.

12-16 The P-CSCF sends the SIP 183 Session Progress response to the previous hop. The I-CSCF is included.

Figure 3.6: Terminating Procedure (part 1)

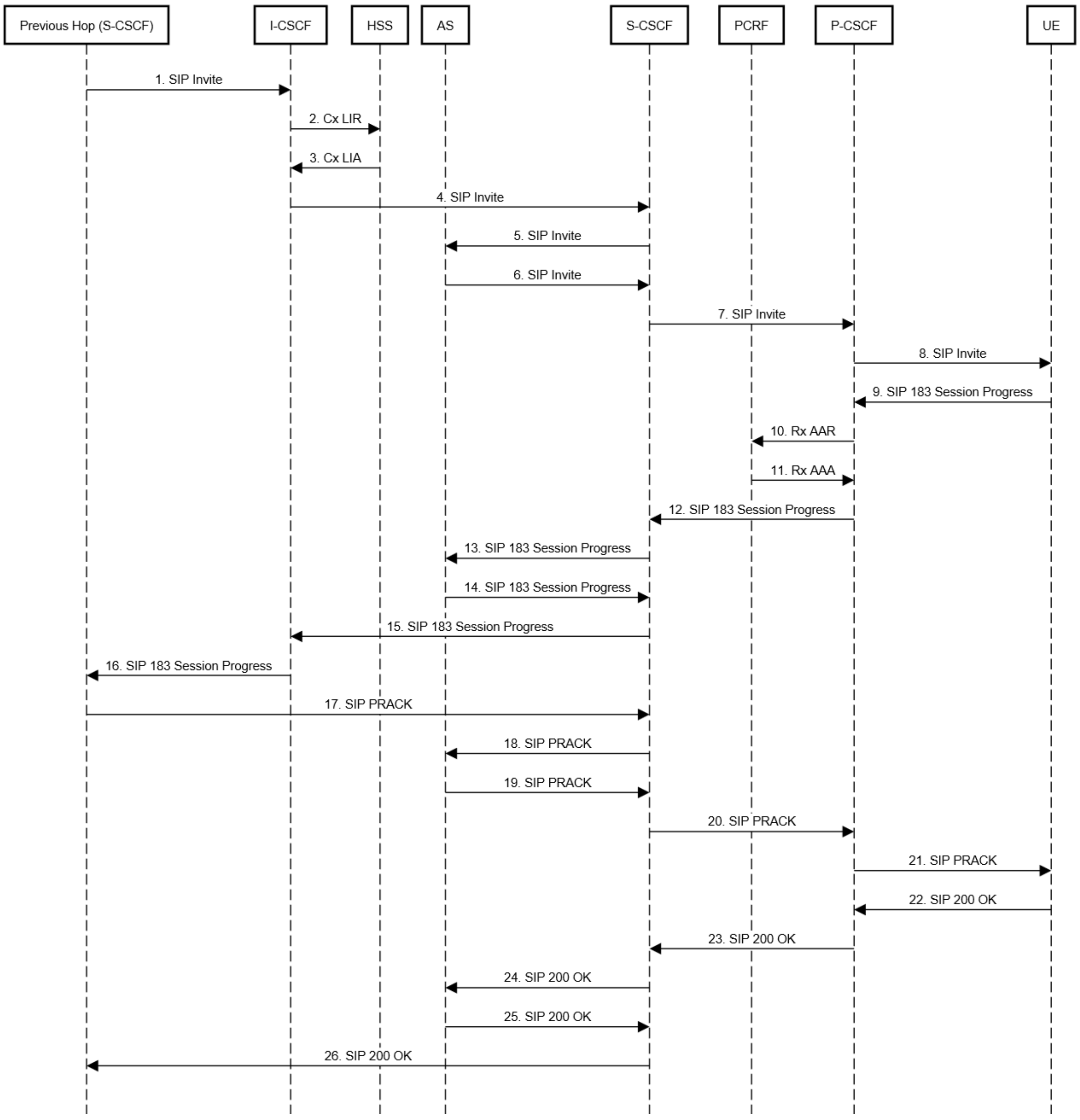
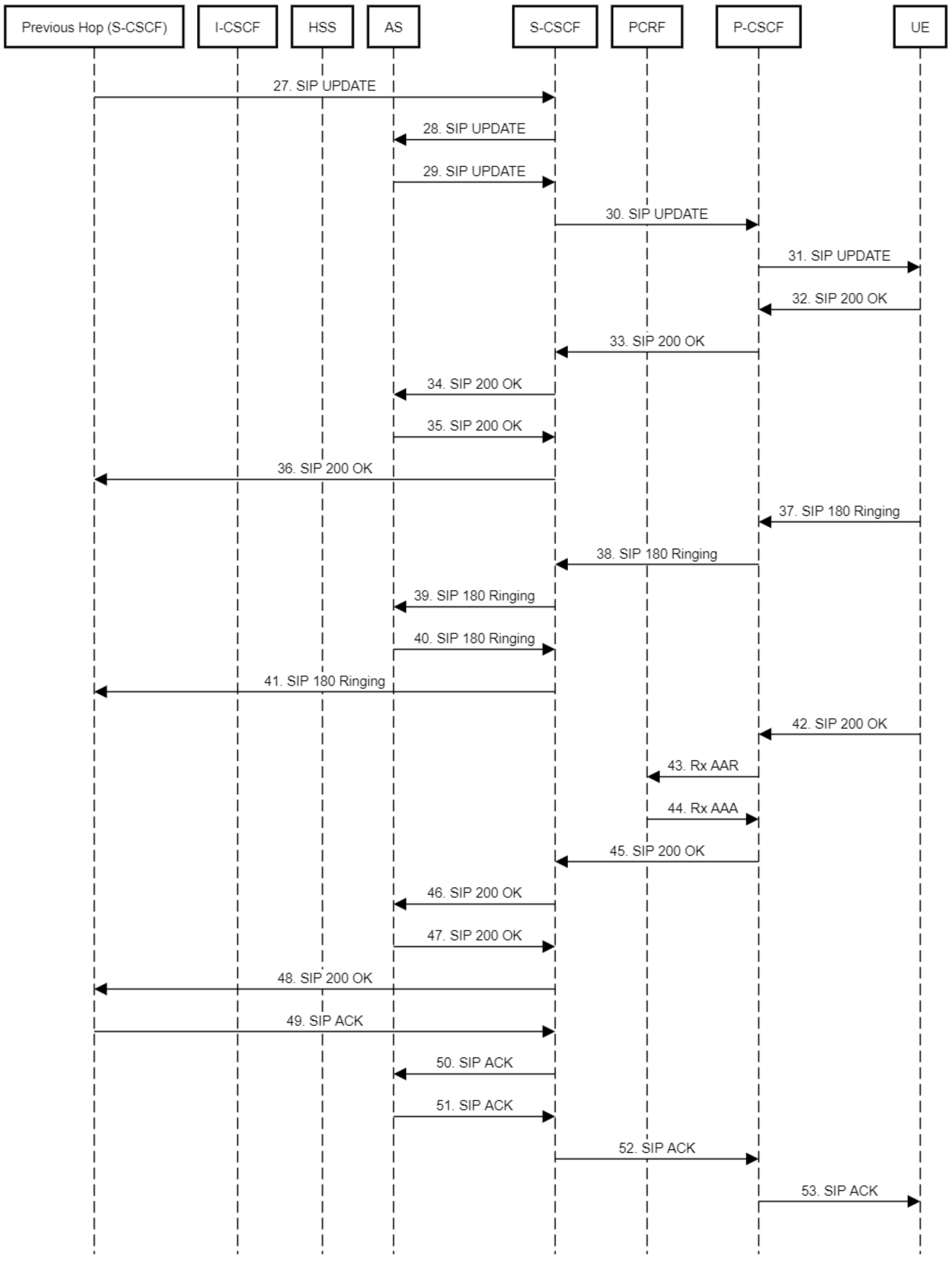


Figure 3.7: Terminating Procedure (part 2)



17-21 The previous hop sends to the UE a SIP PRACK request to acknowledge the SIP 183 response. This message is not sent via the I-CSCF.

22-26 The UE sends back a SIP 200 OK message to acknowledge the reception of the SIP PRACK.

27-31 The calling party sends a SIP UPDATE request with an SDP offer indicating that resources are available.

32-36 The UE sends back a SIP 200 OK to acknowledge the SIP UPDATE request. The SDP offer, which is included in this message, indicates that resources are available.

37-41 The UE sends a SIP 180 Ringing response.

42 When the called user answers the call, the UE sends a SIP 200 OK to the P-CSCF.

43-44 The P-CSCF sends media information to the PCRF to allow the connection in both directions. It also indicates that this information is final.

45-48 The P-CSCF sends the SIP 200 OK to the calling user.

49-53 The calling user sends back a SIP ACK.

3.5. Media Plane

Once the call is established, the media streams can be transmitted between the calling and called parties. This communication goes through the IMS network via the IMS-AGW (IMS Application Gateway) and TrGW (Transition Gateway). The IMS-AGW is located in the border of the IMS network that is facing the access Network. The P-CSCF, via the Iq reference point, interacts with the IMS-AGW for media control. Functions like media flow security, encryption and media transcoding are performed by the IMS-AGW. On the other hand, the TrGW is located in the border of the IMS network that is facing other IP/IMS networks. It is connected to the IBCF via the Ix reference point. Functions like media transcoding, media flow security and filtering are performed by this entity.

4 WebRTC in Mobile Networks

WebRTC is considered by communication network vendors and operators as key technology to enhance their existing voice services by adding additional data stream to the communication. It also enables the creation of new audio and video services. Opposite to current voice solutions such as VoLTE (Voice over LTE) or Wifi Calling, cell phones with a pre-installed Web browser are already WebRTC enabled devices. This is very attractive for mobile operators since new communication capabilities can be massively adopted by users. There is no need to change their end devices to start using WebRTC services. Operators can use their IMS network to integrate WebRTC into their real-time communication framework. It is important to highlight at this point that a user can access the IMS network in multiple ways. For instance, via their cell phones in a VoLTE call, via a WiFi connection in WiFi calling, or in a wired network via application gateways.

The following sections present the integration status of WebRTC in mobile networks. It begins by describing a terminating call model for the integration of WebRTC in the IMS network. Then, an overview of 3GPP standards related to WebRTC is presented. Finally, WebRTC commercial products implemented by Ericsson are described.

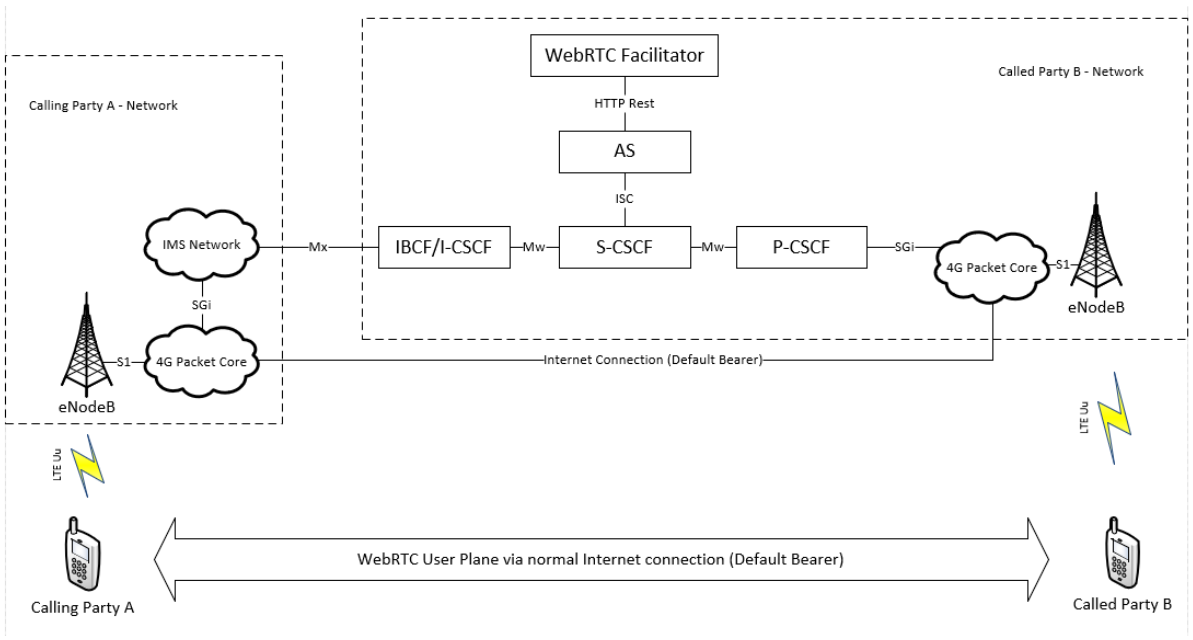
4.1. Over The Top (OTT) Service Model

In this model, WebRTC is considered a service running over the top (OTT) of IMS. The service is triggered by IMS, but WebRTC signaling messages and user data are transmitted via the standard Internet connection. Figure 4.1 presents an overview of this model. This model only applies for terminating call scenarios.

This approach is mainly used to enhance existing voice services. For instance, when a user initiates a VoLTE call, the initial SIP INVITE message will go via the P-CSCF towards the S-CSCF. From there, if the user has WebRTC capabilities activated, an Application Server (AS) can be triggered. This AS will then contact a WebRTC facilitator that will take care of establishing the WebRTC video call. The AS will send back the SIP INVITE message to the S-CSCF to continue with the establishment of the VoLTE call.

In order to establish the WebRTC video call, the WebRTC facilitator needs to inform both, calling and called parties that it is possible to add a video stream to the communication. Both parties can then download a web page with embedded JavaScript code that establishes the WebRTC video communication. A possible solution to accomplish this is by sending a SMS from the WebRTC facilitator towards both parties. This SMS will contain a link to a web page that presents the interface to manage a video WebRTC call. At the end, the VoLTE call will handle the voice communication and WebRTC will be in charge of the video communication. The release of the call will be dictated by the VoLTE call. The S-CSCF must intercept the SIP BYE message and send it to the AS. The AS will ask the WebRTC facilitator to end the

Figure 4.1: Over The Top (OTT) Service Model Architecture



WebRTC video call if this is active.

The main disadvantage of this approach is that the WebRTC video stream will be transmitted via the normal Internet connection (default bearer). This implies that this stream could be affected by competing traffic and channel conditions of the access network. In order to have a reliable connection, specially in mobile applications (e.g. a call inside an ambulance), it is necessary to assign a better QoS target to this video stream. A proposed solution is to create an Rx interface in the WebRTC facilitator towards the Policy and Charging Rules Function (PCRF) to request a dedicated bearer [44]. Another issue with this model is the fact that users must manually access the link contained in the SMS. Mechanisms to automate this process should be implemented. A possible solution could be the provision of a native application on the cell phones that automatically launches once the SMS is received.

4.2. WebRTC in 3GPP Standards

The 3GPP specifies the integration of WebRTC in the IMS network in the annex U of 3GPP TS 23.228 and in TS 24.371 [38][45]. The proposed architecture is based on the following assumptions.

- The codecs used by WebRTC to access IMS must not follow the ones specified by the IETF, but instead the codecs described in TS 26.114 [45] shall be used. This standard specifies for audio the codecs AMR speech, AMR-WB and EVS, while for video the codecs H.264 and H.265 are specified. Web browsers such as Chrome and Firefox implement Opus as the audio codec, and VP8 and H.264 as the video codecs (VP8 is the default video codec). 3GPP mentions that the enhanced IMS-AGW (eIMS-AGW) must be able to perform any required transconding.

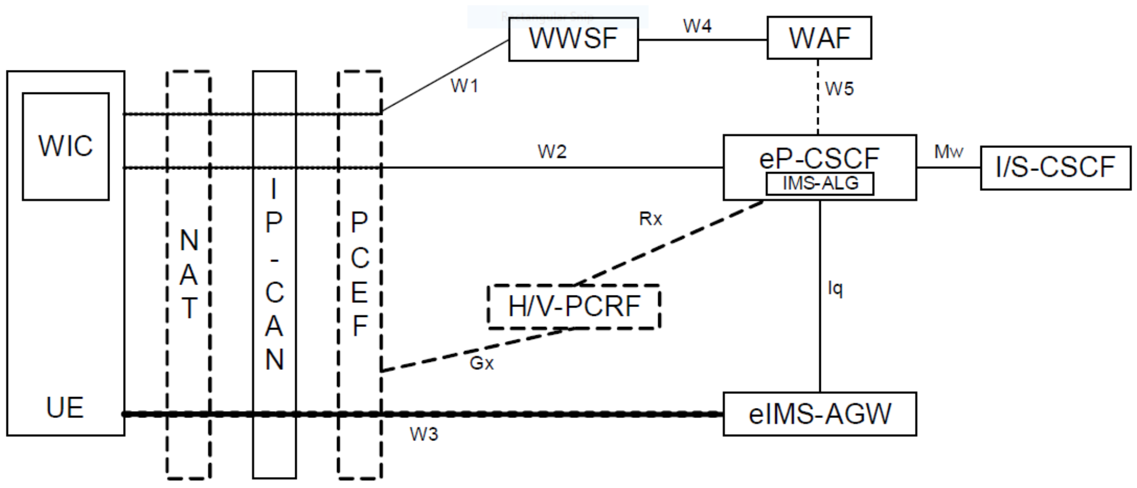
- The protocol for the signaling path between the UE and IMS is not described. Implementations are free to choose any protocol.

- SDP offer/answer exchange must be used for the media negotiation.

- The proposed architecture does not support media multiplexing nor specifies an entity able to perform demultiplexing of the streams. This means that the data channel and audio/video streams must not be sent within the same flow. This imposes a big limitation since the WebRTC Javascript API does not have an option to easily send the streams separately. In fact, media multiplexing is a key feature of WebRTC as mentioned in chapter 2. To fully comply with this 3GPP specification, the demultiplexing of streams must be performed at the client's side. We think that this restriction is with the aim of giving operators the ability to handle streams separately for charging or QoS provision.
- The WebRTC IMS Client (WIC) must not access the ISIM/USIM in the UICC contained in the UE.
- It should be possible to identify the WebRTC flows to provide QoS via PCC functions. The standards do not describe how this should be done.

Figure 4.2, presents the WebRTC IMS architecture proposed by the 3GPP [38]. In this architecture, the eP-CSCF and eIMS-AGW are existing IMS entities which have been enhanced to support WebRTC. The WIC is the application (generally a web browser) in the UE that uses WebRTC protocols. The dashed elements and interfaces are optional. The PCEF and PCRF are used when QoS needs to be provided to the real-time streams. The NAT is part of non-cellular access to IMS. The WebRTC Web Server Function (WWSF) provides the WIC with the web page and Javascript application needed to access IMS. Finally, the WebRTC Authorisation Function (WAF) authorizes users to access IMS when the conventional IMS registration procedure is not used.

Figure 4.2: WebRTC IMS Architecture [38]



4.2.1. Functional entities

As we mentioned above, the WIC is the WebRTC application located in the UE. This application interacts with the IMS architecture in order to establish a real-time communication with a peer (WebRTC or non-WebRTC). It also communicates with the WWSF to download the web page used as interface and the JS code to establish the WebRTC communication. The WWSF is also in charge of allocating IMS identities to WICs and can be located either in the operator network or in a third-party network.

The eP-CSCF (P-CSCF enhanced for WebRTC) is a conventional P-CSCF that has been enhanced to support WebRTC. It must support the protocol used by the WIC for the signaling path (W2 interface) and be able to translate this protocol to SIP (Mw interface). When the

WIC registers its IMPU via conventional IMS authentication, the eP-CSCF must transparently relay the registration messages between WIC and IMS. In case, the WWSF or WebRTC Authorization Function (WAF) authorize the users, the eP-CSCF must check if the WWSF is indeed an authorized entity to provide IMS identities. As we mentioned previously, the eP-CSCF must ensure that the RTP streams are not multiplexed onto the same port. It also interacts with the eIMS-AGW (enhanced IMS-AGW) to control the media plane. The standard mentions that the new WebRTC functions of the P-CSCF might be also performed by a separate entity. This will be addressed in future 3GPP releases.

The eIMS-AGW is a conventional IMS-AGW that has been enhanced to support a WebRTC communication. It must support media security for protocols specific to WebRTC and the DTLS-SRTP key exchange mechanism for SRTP streams. It must also provide NAT traversal support including ICE and transcoding of audio/video codecs if needed.

4.2.2. Reference Points

The description of the WebRTC reference points in figure 4.2 is presented below.

- W1 is the interface between WIC and WWSF. The HTTPs protocol is normally used to access the web page that provides the WebRTC interface and JS code.
- W2 is the signaling path between WIC and eP-CSCF. The protocol is not specified by the 3GPP.
- Iq is the conventional reference point between the P-CSCF and IMS-AGW that has been enhanced to support WebRTC media control functions.
- W3 is the media path between WIC and eP-CSCF. Protocols specific to WebRTC are used on this interface.
- W4 is the signaling interface between WWSF and WAF. WWSF retrieves the authorization tokens from WAF to derive the user's identities when the conventional IMS authentication is not used. The protocol is not specified.
- W5 is the optional interface between eP-CSCF and WAF. The protocol is not specified.

4.2.3. Registration

There exist 3 registration procedures for the proposed WebRTC IMS architecture. These procedures are presented below.

WIC Registration of Individual IMPU/IMPI Using IMS Authentication

In this case, the IMPU and IMPI are obtained by the WIC via unspecified means. For instance, these identities might be provided after visiting a secure web site of the IMS operator. The WIC downloads the web page from the WWSF and then establishes a connection towards the eP-CSCF. Subsequent registration messages are exchanged between WIC and IMS.

WIC Registration of Individual IMPU/IMPI Based on Web Authentication

The WIC initiates a HTTPs connection towards the WWSF to download the web page. The WWSF authenticates the user via common web authentication procedures (i.e. user and password). The WWSF then retrieves the IMS identities and obtains a security token from the WAF. This information is sent back to the WIC. The WWSF may decide to include or not the IMS identities. After this, the WIC establishes a connection towards the eP-CSCF and sends a registration message including the security token and IMS identities if these were sent by the WWSF. The eP-CSCF validates the security token and then obtains the IMS identities which are used to initiate an IMS registration towards the S-CSCF using the Trusted Node Authentication (TNA) procedure.

WIC Registration of Individual IMPU/IMPI from a pool of IMPUs/IMPIs

In this method, the WWSF is allocated with a pool of IMS subscriptions each one having a single IMPI and multiple IMPUs. Users are assigned with an IMPU and IMPI during registration. The WWSF might be located in a third party network. The registration procedure is similar to the one previously described. The only difference is that the WWSF might decide not to authenticate the user. Unauthenticated users are anonymous to the third party network.

4.2.4. End-to-End Session Procedures

Originating and terminating session procedures follow standard IMS procedures.

4.2.5. Media Plane Optimization

The standard also specifies that when a real-time communication is established between two WebRTC clients, the media plane is not modified by the eIMS-AGW. This means that the WebRTC protocols are transparently relayed in the IMS network. An exception occurs when the operator desires to perform Lawful Interception (LI).

4.3. Ericsson Commercial Products

Ericsson, as one of the leading telecommunication vendors in the world, sees WebRTC as the key technology to enhance their communication portfolio. As a prove of this, solutions based on WebRTC have been developed. A brief description of the main features of these solutions are presented below.

4.3.1. Ericsson Web Communication Gateway (WCG)

The Web Communication Gateway (WCG) is a product that enables users to establish real-time communication services via the IMS network. Users can access IMS with devices connected to the Internet via cable, Wi-Fi or cellular. WCG handles the WebRTC signaling communication between users and IMS. The media plane goes directly between users and IMS-AGWs. Compared to the specifications of the 3GPP presented in the previous section, the functions performed by the WCG are similar to the ones performed by the eP-CSCF and WAF. A key difference, however, is that the WCG do not block the exchange of SDP messages to multiplex real-time streams onto the same port. Note that to be able to establish a WebRTC communication, the existing P-CSCF and IMS-AGWs deployed on the operator's network must have been enhanced to support protocols specific to WebRTC, namely SRTP and the DTLS-SRTP key exchange mechanism. Ericsson P-CSCF and IMS-AGW products already support this new functionality.

One of the modes of operation of WCG is called Web Access. In this mode of operation, the WCG acts as a HTTP/REST and Websocket server towards the web client, and as a UE towards IMS. In other words, it acts as a HTTP/SIP gateway between web client users and IMS. Via RESTful APIs, users are able to perform conventional registration and end-to-end session procedures. Users must know the URI of the WCG and IMS identities. This information might be provided by a Web server (WWSF in 3GPP) managed by the operator or a third party company.

4.3.2. Ericsson Contextual Communication Cloud (EC3)

The Ericsson Contextual Communication Cloud (EC3) is a service based on WebRTC that allows users to improve their communication services. It is an enhanced WebRTC signalling server that provides users with a Javascript based Web SDK that can be used to develop WebRTC services with the support of additional features such as: screen sharing, media recording, storage options, co-browsing and co-editing. This SDK uses mainly WebRTC API and Matrix (Synapse).

Ericsson has developed prototypes that enhance a VoLTE call with WebRTC by triggering an application server from the S-CSCF when this receives a VoLTE SIP Invite message. The application server communicates with EC3 and a message is sent to the called and calling parties to wake up a WebRTC application previously installed in their phones. Via this application, users can enrich their audio call with all features offered by EC3. It is important to note that the media communication is not transported via the IMS nor the dedicated VoLTE bearer. This communication goes via the 4G default bearer (Internet connection).

5 Experimental Setup

This chapter describes the experiments used to evaluate the performance of GCC in wired, wireless and mobile networks. The experimental setup follows the evaluation guidelines for RMCAT congestion control algorithms defined by IETF in [46]. The main objective is to determine whether GCC is able to fulfill the requirements for real-time multimedia communication [4]. The experiments differ per access technology and are run in controlled networks (testbeds).

5.1. General Setup

This section presents the hardware, tools and general settings used in the experiments.

5.1.1. Hardware

The hardware consists of four laptops with the following specifications.

- PC1 - Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz, 8G RAM, CentOS Linux 7
- PC2 - Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, 8G RAM, CentOS Linux 7
- PC3 - Intel(R) Core(TM) i5-4310U CPU @ 2.00GHz, 8G RAM, CentOS Linux 7
- PC4 - Intel(R) Core(TM) i5-4310U CPU @ 2.00GHz, 8G RAM, CentOS Linux 7

5.1.2. Web Browser

Google Chrome is the browser used for the WebRTC communication. PC1 has a Web Server deployed with Node.js that provides HTML pages with embedded WebRTC Javascript API code. This server also handles the WebRTC signaling traffic between the laptops. Google Chrome allows to feed a Y4M file instead of a live camera input. This is used to inject one of the video sequences recommended in [46]. By default, Chrome uses the video encoder VP8. All experiments use this video encoder. As a side note, we want to mention that the worldwide browser market share according to [47], has Chrome in the first place with 63.34% followed by Safari and Firefox with 15.06% and 4.48% respectively.

5.1.3. RTP Media Source

All experiments use the same video stream for the WebRTC communication. The chosen video stream is ”Foreman” which is one of the two video streams selected in [46]. This video stream has a frame resolution of 1280x720 pixels and a frame rate of 60 fps. This video is categorized as a High Definition (HD) video.

5.1.4. Cross Traffic

Iperf is used to generate long-lived flows over TCP. For experiments requiring bidirectional TCP flows (one uplink and one downlink), two parallel flows are created. The default congestion control algorithm of TCP in CentOS Linux 7 is Cubic.

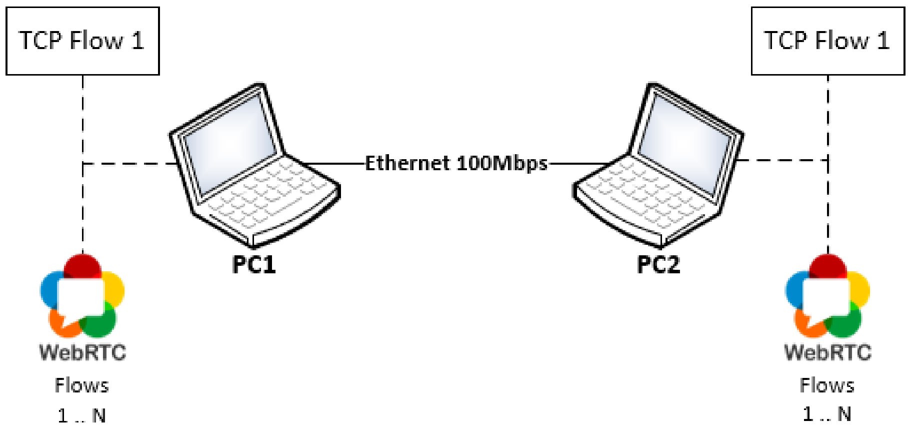
5.2. Wired Network

The evaluation of GCC in a wired network covers some of the basic usage scenarios defined in [8]. The following subsections present the specific setup for this type of network, as well as, the description of the experiments.

5.2.1. Testbed Topology

Figure 5.1 presents the general testbed topology used for the experiments. Laptops PC1 and PC2, are directly connected via their 100 Mbps Ethernet interfaces. Each experiment specifies the number and direction of the RTP flows and also whether a TCP flow is present or not. We denominate the traffic from PC1 to PC2 as forward and the traffic from PC2 to PC1 as backward.

Figure 5.1: Testbed Topology - Wired Network



5.2.2. Path Characteristics

The path characteristics used in the experiments are based on the recommendations found in [46] and the detailed study of the edge network properties found in [48]. This latter study shows that the link capacity is mainly limited in the uplink direction. We have chosen to perform measurements with values between 1 Mbps and 4 Mbps. It is important to mention at this point that the VP8 video encoder limits the sending bit rate to 2.5 Mbps. With respect to latency, typical values were found between 100 ms and 200 ms. Finally, we have chosen tail drop as the queue type due to its prevalent nature on networks. The buffer length is set to the nominal value of 300 ms.

5.2.3. Traffic Control

In this project, Linux Traffic Control (TC) and Netem are used to simulate the properties of WAN links as described in [49]. A Token Bucket Filter (TBF) is used to limit the bandwidth of the Ethernet links.

5.2.4. Evaluation Metrics

The following metrics are used to evaluate the performance of GCC. These metrics are generated from the statistics API of WebRTC which are collected every 500 ms.

- GCC bit rate R $[Mbps]$ - Sending bit rate.
- Bandwidth utilization B [%] - Ratio between average link capacity and available capacity,
- Packet loss L [%] - Ratio between packets lost and packets sent.
- Queuing delay QD $[ms]$ - Measured RTT minus the minimum RTT obtained during the whole experiment.

- Frame Rate FR $[fps]$ - Frame rate at the receiving side.
- Frame width and height FW $[px]$ and FH $[px]$ - Frame width and height at the receiving side.
- Converge time C $[ms]$ - Time needed to reach a stable sending rate.

5.2.5. Unconstrained Capacity with a Single Flow - Experiment 1

The goal of this experiment is to obtain performance metrics of GCC when there are no constrains on the link that connects the two laptops. The results will show how the encoders and GCC behaves when there is a link with high bandwidth and low latency.

Testbed Topology

A single RTP flow and zero TCP flows. See 5.2.1.

Testbed Attributes

- Duration: 120 s
- Path Characteristics:
  - Path Direction: Forward
  - Reference Bottleneck Capacity: 100 Mbps
  - One-way propagation delay: 0.5 ms
  - Bottleneck queue type: Tail drop
  - Path loss ratio: 0 %
- Application-related:
  - Media traffic: as described in 5.1.3
  - Competing traffic: zero (0)

5.2.6. Variable Available Capacity with a Single Flow - Experiment 2

In this experiment, the link capacity between the two peers varies over time. The constraint in capacity emulates the presence of intermediate bottlenecks. In real networks, the variation of path capacity might be due to a routing change or the abrupt arrival/departure of loss-based traffic. The main goal of this experiment is to determine if GCC can adapt its real-time flow while maintaining at the same time low end-to-end delay.

Expected Behavior

GCC detects the variation in capacity and adapts its sending bit rate accordingly. The converge time should be small and the sending rate should be stable when the bit rate is close to the available capacity.

Testbed Topology

A single RTP flow and zero TCP flows. See 5.2.1.

Testbed Attributes

- Duration: 160 s
- Path Characteristics:
  - Path Direction: Forward
  - Reference Bottleneck Capacity: [1, 2, 3] Mbps
  - One-way propagation delay: [25, 50, 100] ms
  - Maximum end-to-end jitter: 0 ms
  - Bottleneck queue type: Tail drop

  - Bottleneck queue size: 300 ms
  - Path loss ratio: 0 %

- Application-related:
  - Media traffic: as described in 5.1.3
  - Competing traffic: zero (0)

- Link variation pattern:
  1. Time: 0 s - Capacity: 1 Mbps
  2. Time: 40 s - Capacity: 3 Mbps
  3. Time: 80 s - Capacity: 1 Mbps
  4. Time: 120 s - Capacity: 2 Mbps

5.2.7. Competing Media Flows with Same Congestion Control Algorithm - Experiment 3

The aim of this experiment is to evaluate the capacity of GCC to share the channel bandwidth when multiple real-time flows are present in the same bottleneck. In real networks, it is very likely that media flows are transported via the same bottleneck to reach a common endpoint.

Expected Behavior

GCC should be able to detect the presence of other media flows and decrease its sending rate in order to fairly use the capacity of the channel. Ideally, the full capacity of the channel should be equally divided among the flows. An incoming media flow should not increase the queuing delay or produce packet loss to an existing media flow.

Testbed Topology

Three RTP flows and zero TCP flows. See 5.2.1.

Testbed Attributes

- Duration: 420 s

- Path Characteristics:
  - Path Direction: Forward
  - Reference Bottleneck Capacity: 4 Mbps
  - One-way propagation delay: 100 ms
  - Maximum end-to-end jitter: 0 ms
  - Bottleneck queue type: Tail drop
  - Bottleneck queue size: 300 ms
  - Path loss ratio: 0 %

- Application-related:
  - Media traffic: as described in 5.1.3
  - Competing traffic: zero (0)

- RTP Flows:
  1. Time: 0 s - Flow 1
  2. Time: 60 s - Flows 1 and 2
  3. Time: 240 s - Flows 1, 2 and 3

5.2.8. Media Flow Competing with a Long-lived TCP Flow - Experiment 4

This experiment evaluates the performance of GCC when a long-lived TCP flow is transmitted over the same bottleneck. This emulates the scenario where a multimedia flow competes against a very large file download. Since the TCP flow can fill up the buffer capacity of an equipment located in the edge network, the performance with a bottleneck queue size of 1000 ms is also analyzed.

Expected Behavior

GCC must not be starved and collapsed by the competing TCP flow. It should adjust its sending bit rate to fairly compete against this loss-based traffic.

Testbed Topology

A single RTP flow and one TCP flow. See 5.2.1.

Testbed Attributes

- Duration: 240 s
- Path Characteristics:
  - Path Direction: Forward
  - Reference Bottleneck Capacity: 2 Mbps
  - One-way propagation delay: 100 ms
  - Maximum end-to-end jitter: 0 ms
  - Bottleneck queue type: Tail drop
  - Bottleneck queue size: [300, 1000] ms
  - Path loss ratio: 0 %
- Application-related:
  - Media traffic: as described in 5.1.3
  - Competing traffic: one (1)
- TCP/RTP Flows:
  1. Time: 0 s - Only TCP flow
  2. Time: 30 s - Both TCP and RTP flows

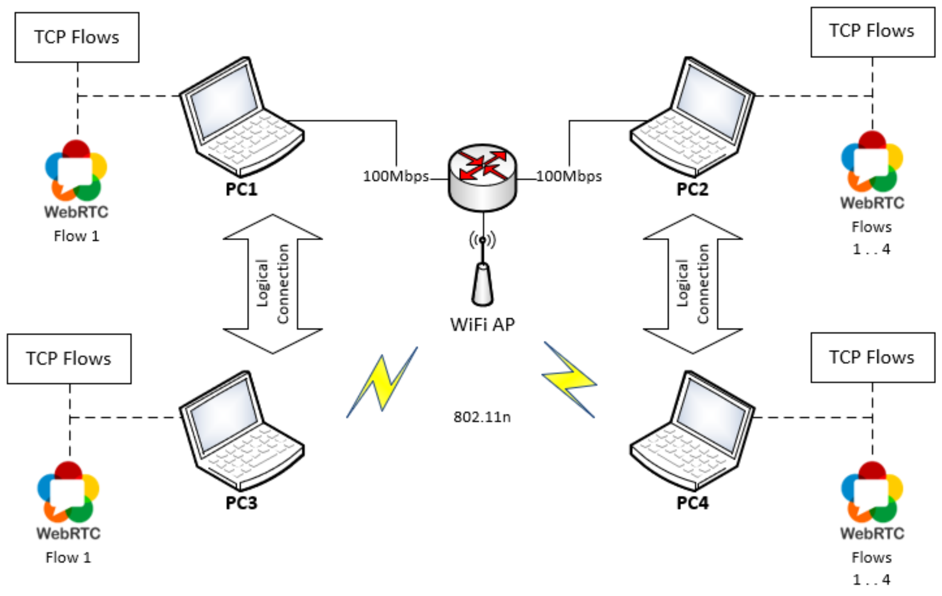
5.3. WiFi Network

The evaluation of GCC in WiFi follows partially the use cases defined in [50] and combines them with the methodology found in [8]. It was not possible to fully follow [50] since there was a limitation in hardware resources that prevented the addition of WebRTC flows. The next subsections present the specific setup for this network and the description of the experiments.

5.3.1. Testbed Topology

Figure 5.2 presents the general testbed topology used for the experiments. Laptops PC1 and PC2 are connected to the WiFi access point (AP) via their 100 Mbps Ethernet interfaces. On the other hand, laptops PC3 and PC4 connect to the AP via their WiFi interfaces. The topology presents two logical connections for the WebRTC and TCP flows. These connections are between PC1 and PC3, and between PC2 and PC4. The WebRTC flow established between PC1 and PC3 is used to evaluate the performance of GCC under different network conditions. The connection between PC2 and PC4 is used with the aim of congesting the radio channel. A total of 4 WebRTC flows and 1 TCP flow is used for this purpose.

Figure 5.2: Testbed Topology - WiFi Network



5.3.2. WiFi Access Point

The access point used in the experiments is a ZTE model H369A. The evaluation of GCC is performed using the 802.11n technology in the 2.4 GHz band and channel 11.

5.3.3. Locations

In order to be able to evaluate GCC under different network conditions, PC3 and PC4 are placed in two locations, namely location A and B. Location A is located close to the AP in a room with no obstacles in between. It was observed that this location presented MCS (Modulation Coding Scheme) codes indicating theorical data rates above 100 Mbps. On the other hand, location B is away from the AP and there are many obstacles including walls between this location and the AP. MCS codes for this location indicated theorical data rates between 25 and 40 Mbps. All experiments are run in three different scenarios which are described below.

- Scenario 1: PC3 and PC4 placed in location A.
- Scenario 2: PC3 and PC4 placed in location B.
- Scenario 3: PC3 placed in location B and PC4 placed in location A.

5.3.4. Evaluation Metrics

The following metrics are used to evaluate the performance of GCC. These metrics are generated from the statistics API of WebRTC which are collected every 500 ms.

- GCC bit rate R $[Mbps]$ - Sending bit rate.
- Packet loss L [%] - Ratio between packets lost and packets sent.
- Queuing delay QD $[ms]$ - Measured RTT minus the minimum RTT obtained during the whole experiment.
- Frame Rate FR $[fps]$ - Frame rate at the receiving side.
- Frame width and height $FW\ [px]\ and\ FH\ [px]$ - Frame width and height at the receiving side.
- Bandwidth Limited Resolution BLR - WebRTC indicator for limited resolution due to bandwidth.

5.3.5. Competing Media Flows Sharing the Wireless Uplink - Experiment 1

The aim of this experiment is to evaluate the capacity of GCC to adjust its bitrate based on channel conditions and the presence of other GCC flows in the uplink direction. Multiple GCC flows transmitting simultaneously in the uplink direction increases the channel contention and number of collisions.

Expected Behavior

GCC should be able to detect congestion in the channel due to competing GCC traffic and adapt its bitrate to prevent packet loss. It should maintain low and stable delay during the congestion period. The frame rate must be stable and the resolution of the video stream should not be considerably decreased.

Testbed Topology

PC1/PC3 - A single GCC flow (uplink), PC2/PC4 - 4 GCC flows (uplink). See 5.3.1.

Testbed Attributes

- Duration: 180 s
- Path Characteristics:
  - Radio Configuration: 802.11n, 2.4GHz, Channel 11
  - Locations: Scenarios 1 and 2. See 5.3.3.
- Application-related:
  - Media traffic: as described in 5.1.3
  - Competing traffic: zero (0)
- Link variation pattern:
  1. Time: 0 s - PC1/PC3: 1 GCC flow (uplink)
  2. Time: 60 s - PC1/PC3: 1 GCC flow (uplink), PC2/PC4: 2 GCC flow (uplink)
  3. Time: 120 s - PC1/PC3: 1 GCC flow (uplink), PC2/PC4: 4 GCC flow (uplink)

5.3.6. Competing Media Flows Sharing the Wireless Downlink - Experiment 2

This experiment evaluates the capacity of GCC to adjust its bitrate based on channel conditions and the presence of other GCC flows in the downlink direction. The test emulates a user who is receiving a real-time video in a WiFi network where other users are also using real-time services.

Expected Behavior

GCC should be able to detect congestion in the channel due to competing GCC traffic and adapt its bitrate to prevent packet loss. It should maintain low and stable delay during the congestion period. The frame rate must be stable and the resolution of the video stream should not be considerably decreased. The performance in terms of throughput should be better than in the previous experiment (uplink).

Testbed Topology

PC1/PC3 - A single GCC flow (downlink), PC2/PC4 - 4 GCC flows (downlink). See 5.3.1.

Testbed Attributes

- Duration: 180 s
- Path Characteristics:
  - Radio Configuration: 802.11n, 2.4GHz, Channel 11
  - Locations: Scenarios 1 and 2. See 5.3.3.

- Application-related:
    - Media traffic: as described in 5.1.3
    - Competing traffic: zero (0)
- Link variation pattern:
    1. Time: 0 s - PC1/PC3: 1 GCC flow (downlink)
    2. Time: 60 s - PC1/PC3: 1 GCC flow (downlink), PC2/PC4: 2 GCC flow (downlink)
    3. Time: 120 s - PC1/PC3: 1 GCC flow (downlink), PC2/PC4: 4 GCC flow (downlink)

5.3.7. Competing Media Flows in a Bidirectional Transmission - Experiment 3

The purpose of this experiment is to determine the quality of a WebRTC call in a WiFi network where other users have also established WebRTC calls. Since the radio channel is shared by the uplink and downlink transmissions, the channel contention and collisions are higher than in the previous experiments.

Expected Behavior

GCC should be able to detect congestion in the channel due to competing GCC traffic and adapt its bitrate to prevent packet loss. It should maintain low and stable delay during the congestion period. The frame rate must be stable and the resolution of the video stream should not be considerably decreased. The uplink and downlink should have similar performance and one cannot starve or be starved by the other.

Testbed Topology

PC1/PC3 - A single GCC flow (bidirectional), PC2/PC4 - 4 GCC flows (bidirectional). See 5.3.1.

Testbed Attributes

- Duration: 180 s
- Path Characteristics:
    - Radio Configuration: 802.11n, 2.4GHz, Channel 11
    - Locations: Scenarios 1, 2 and 3. See 5.3.3.
- Application-related:
    - Media traffic: as described in 5.1.3
    - Competing traffic: zero (0)
- Link variation pattern:
    1. Time: 0 s - PC1/PC3: 1 GCC flow (bidirectional)
    2. Time: 60 s - PC1/PC3: 1 GCC flow (bidirectional), PC2/PC4: 2 GCC flow (bidirectional)
    3. Time: 120 s - PC1/PC3: 1 GCC flow (bidirectional), PC2/PC4: 4 GCC flow (bidirectional)

5.3.8. Media Flow Competing with a Long-lived TCP Flow - Experiment 4

This experiment emulates a user who has established a WebRTC call in a WiFi network where another user is downloading a large file. The aim of this experiment is to evaluate the performance of GCC in the presence of a TCP flow which can increase the packet loss and delay in the network. The ability of GCC to increase its bitrate after the TCP flow stops transmitting is also evaluated.

Expected Behavior

GCC should be able to detect congestion in the channel due to a competing TCP traffic and adapt its bitrate to reduce packet loss. The delay should not largely increase by the presence of the TCP flow. The frame rate must be stable and the resolution of the video stream should not be considerably decreased. The GCC flow must not starve or be starved by the TCP flow. After the TCP flow stops transmitting, GCC should be able to increase its bitrate shortly.

Testbed Topology

PC1/PC3 - A single GCC flow (bidirectional) and a TCP flow (bidirectional), PC2/PC4 - a single TCP flow (bidirectional). See 5.3.1.

Testbed Attributes

- Duration: 240 s
- Path Characteristics:
  - Radio Configuration: 802.11n, 2.4GHz, Channel 11
  - Locations: Scenarios 1, 2 and 3. See 5.3.3.
- Application-related:
  - Media traffic: as described in 5.1.3
  - Competing traffic: zero (0)
- Link variation pattern:
  1. Time: 0 s - PC1/PC3: 1 GCC flow (bidirectional)
  2. Time: 60 s - PC1/PC3: 1 GCC flow (bidirectional) and 1 TCP flow (downlink), PC2/PC4: 1 TCP flow (downlink)
  3. Time: 120 s - PC1/PC3: 1 GCC flow (bidirectional) and 1 TCP flow (bidirectional), PC2/PC4: 1 TCP flow (bidirectional)
  4. Time: 180 s - PC1/PC3: 1 GCC flow (bidirectional)

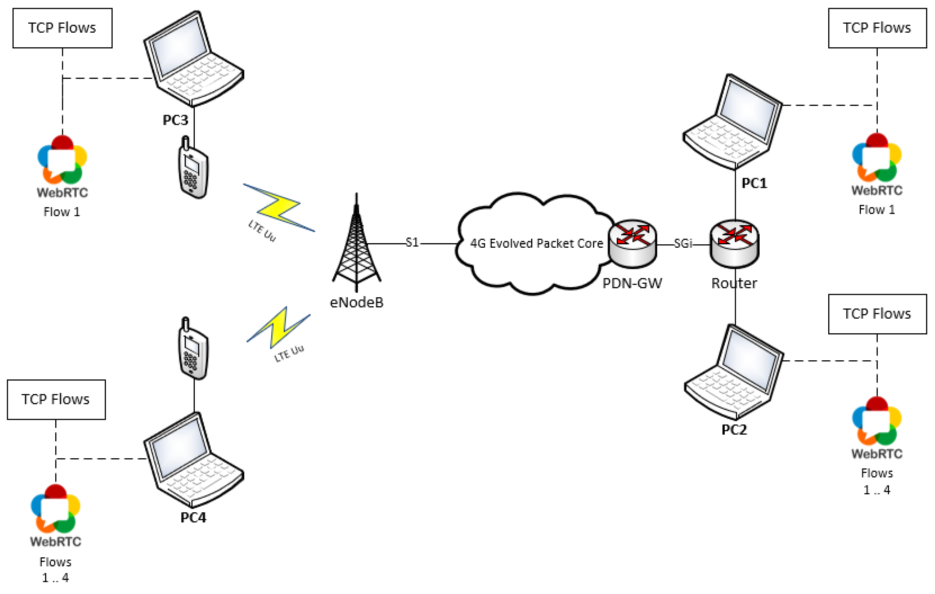
5.4. 4G Network

The evaluation of GCC in a 4G network follows some of the WiFi use cases defined in [50]. This document only defines 4G use cases for simulated networks. Therefore, it was decided to take a similar approach as in WiFi. There was again a limitation in hardware resources that prevented the addition of multiple WebRTC flows. The next subsections present the specific experimental setup for this network and the description of the experiments.

5.4.1. Testbed Topology

The topology used in the experiments is presented in figure 5.3. Laptops PC1 and PC2 are connected to the 4G core network via a switch that is connected to the PDN-GW (Packet Data Network - Gateway). PC3 and PC4 access the network via the 4G wireless connection. For this purpose, two cell phones are connected to these laptops via USB tethering to provide the 4G mobile connection. There is a single 4G outdoor antenna in the network. There exist two logical connections for the WebRTC and TCP flows. These connections are between PC1 and PC3, and between PC2 and PC4. The WebRTC flow established between PC1 and PC3 is used to evaluate the performance of GCC. The connection between PC2 and PC4 is used to congest the radio channel. There are no additional users in the network when the experiments are run.

Figure 5.3: Testbed Topology - 4G Network



5.4.2. Hardware

Besides the 4 laptops that have been previously presented, two cell phones were used to provide the 4G mobile data communication. The specifications of these cell phones are presented below.

- Samsung Galaxy A50 - Model # SM-A505FN - Android version 9
- Samsung Galaxy S8 - Model # SM-G950F - Android version 8.0.0

5.4.3. Radio System

The 4G testbed uses the radio unit RRU 2212 of Ericsson. The 4G radio works in the 1800 MHz band and both, uplink and downlink radio link has 5 MHz of bandwidth. The power transmission is 2 Watt. There is a single directional sector antenna (MIMO 2x2).

5.4.4. Locations

The evaluation of GCC in this network is performed in two scenarios. In the first scenario, laptops PC3 and PC4 are located very close (about 5 meters) to the antenna and have direct line of sight. On the other hand, the second scenario has the two laptops located far (about 100 meters) from the antenna with many obstacles in between them, including trees and a building. Moreover, the position of these laptops was about 90 degrees to the right of the main lobe of the antenna.

5.4.5. Evaluation Metrics

The following metrics are used to evaluate the performance of GCC. These metrics are generated from the statistics API of WebRTC which are collected every 500 ms.

- GCC bit rate R $[Mbps]$ - Sending bit rate.
- Packet loss L $[\%]$ - Ratio between packets lost and packets sent.
- Queuing delay QD $[ms]$ - Measured RTT minus the minimum RTT obtained during the whole experiment.
- Frame Rate FR $[fps]$ - Frame rate at the receiving side.

- Frame width and height $FW\ [px]\ and\ FH\ [px]$ - Frame width and height at the receiving side.
- Bandwidth Limited Resolution BLR - WebRTC indicator for limited resolution due to bandwidth.

5.4.6. Competing Media Flows in a Bidirectional Transmission - Experiment 1

This experiment determines the quality of a bidirectional WebRTC call in a 4G network when the radio channel is being also utilized by other bidirectional WebRTC calls. The results will show how GCC reacts to congestion caused by other GCC flows.

Expected Behavior

GCC should be able to detect congestion in the channel due to competing GCC traffic and adapt its bitrate to prevent packet loss. It should maintain low and stable delay during the congestion period. The frame rate must be stable and the resolution of the video stream should not be considerably decreased.

Testbed Topology

PC1/PC3 - A single GCC flow (bidirectional), PC2/PC4 - 4 GCC flows (bidirectional). See 5.4.1.

Testbed Attributes

- Duration: 180 s
- Path Characteristics:
  - Radio Configuration: as described in 5.4.3.
  - Locations: Scenarios 1 and 2. See 5.4.4.
- Application-related:
  - Media traffic: as described in 5.1.3
  - Competing traffic: zero (0)
- Link variation pattern:
  1. Time: 0 s - PC1/PC3: 1 GCC flow (bidirectional)
  2. Time: 60 s - PC1/PC3: 1 GCC flow (bidirectional), PC2/PC4: 2 GCC flow (bidirectional)
  3. Time: 120 s - PC1/PC3: 1 GCC flow (bidirectional), PC2/PC4: 4 GCC flow (bidirectional)

5.4.7. Media Flow Competing with a Long-lived TCP Flow - Experiment 2

This experiment emulates a user who has established a WebRTC call in a 4G network where another user is downloading a large file. The aim of this experiment is to evaluate the performance of GCC in the presence of a TCP flow which can increase the packet loss and delay in the network.

Expected Behavior

GCC should be able to detect congestion in the channel due to competing TCP traffic and adapt its bitrate to reduce packet loss. The delay should not largely increase by the presence of the TCP flow. The frame rate must be stable and the resolution of the video stream should not be considerably decreased. The GCC flow must not starve or be starved by the TCP flow. After the TCP flow stops transmitting, GCC should be able to increase its bitrate shortly.

Testbed Topology

PC1/PC3 - A single GCC flow (bidirectional) and a single TCP flow (bidirectional), PC2/PC4 - a single TCP flow (bidirectional). See 5.4.1.

Testbed Attributes

- Duration: 240 s
- Path Characteristics:
  - Radio Configuration: as described in 5.4.3.
  - Locations: Scenarios 1 and 2. See 5.4.4.
- Application-related:
  - Media traffic: as described in 5.1.3
  - Competing traffic: zero (0)
- Link variation pattern:
  1. Time: 0 s - PC1/PC3: 1 GCC flow (bidirectional)
  2. Time: 60 s - PC1/PC3: 1 GCC flow (bidirectional), PC2/PC4: 1 TCP flow (bidirectional)
  3. Time: 120 s - PC1/PC3: 1 GCC flow (bidirectional) and 1 TCP flow (bidirectional), PC2/PC4: 1 TCP flow (bidirectional)
  4. Time: 180 s - PC1/PC3: 1 GCC flow (bidirectional)

6 Performance Analysis

This chapter presents the results obtained from the experiments described in chapter 5. The results are analyzed to determine if GCC fulfills the congestion control requirements for real-time media streams defined by IETF [4].

6.1. Congestion Control Requirements

In order to evaluate GCC it is necessary to compare its performance against a set of requirements specified for congestion control algorithms for real-time media flows. The IETF, in [4], has defined these requirements for WebRTC traffic. The requirements relevant for this project are presented below.

1. It should provide, as much as possible, low end-to-end delay while still providing a useful amount of bandwidth. The end-to-end delay should be lower than 100 ms.
   (a) It should provide this low end-to-end delay even in the presence of competing traffic or intermediate bottlenecks.
   (b) It should react fast to the addition or removal of intermediate bottlenecks or changes in the available bandwidth caused by network alterations such as rerouting of traffic. It must prevent, as much as possible, any increase of delay. The IETF expects reaction times of around 1 second.
   (c) It should prevent, as much as possible, the increase of delay due to the presence of a TCP flow which might be saturating the network. It should react fast to the addition or removal of TCP flows.
2. It should be fair to other flows, both real-time flows and TCP flows. It should prioritize a fair share of bandwidth depending on the media types and available bandwidth over achieving, as much as possible, low end-to-end delay.
   (a) Existing flows in a bottleneck should be fair to new flows coming in the network. These flows should be allowed to rapidly increase their bitrate.
3. It should not starve or be starved by TCP flows present in the network.
4. It should adapt, as fast as possible, to initial network conditions. In other words, it should have fast convergence at startup.
5. It should provide a stable bitrate after convergence.

6.2. Wired Network

The following subsections present the results of the experiments performed in a wired network. In this type of network it was possible to modify network conditions in a controlled way for the assessment of GCC.

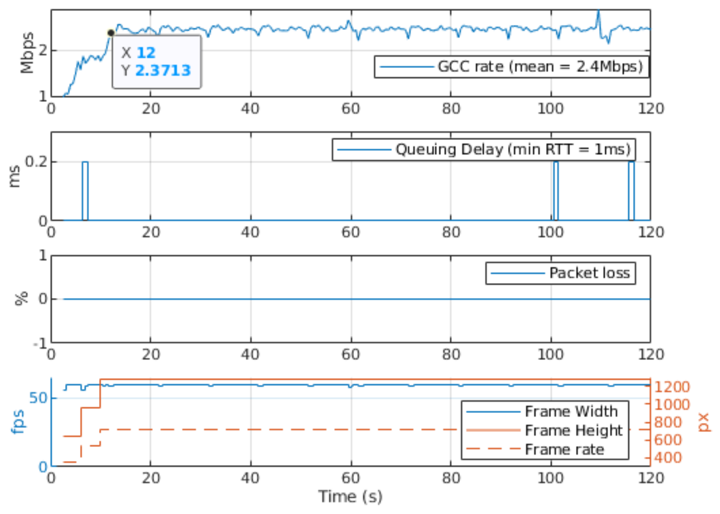


Figure 6.1: Results experiment 1 - Wired network

6.2.1. Unconstrained Capacity with a Single Flow - Experiment 1

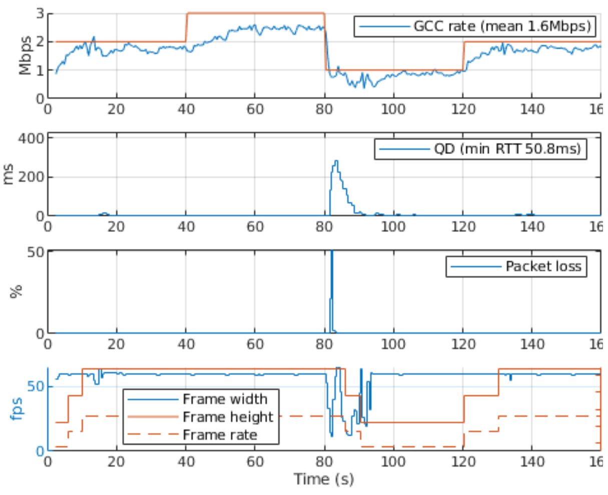
This experiment shows the performance of GCC in a channel with high bandwidth, very low latency and no packet loss. The results, which are presented in figure 6.1, set a benchmark for subsequent experiments.

The upper graph of this figure presents the GCC sending bitrate. It reached an average of 2.4 Mbps. As it was mentioned previously, the VP8 codec imposes an upper limit of 2.5 Mbps in the sending bitrate. It can be observed that this bitrate took a long time to reach its maximum value after the WebRTC call was established. On average, the connection establishment took 1 second and the bitrate was only able to reach its maximum around 12 seconds. This gives us a convergence time equal to 11 seconds. Initially, GCC was increasing its bitrate in multiplicative mode to reach a stable value as soon as possible. When the bitrate was close to 2 Mbps, GCC switched to additive mode. Chapter 2 mentions that GCC switches from multiplicative mode to additive mode when the bitrate is within 3 standard deviations from a previous stable bitrate, but since this increase in bitrate is happening after starting up the WebRTC call, there is not a previous stable bitrate saved. Therefore, this switch at 2 Mbps should correspond to an implementation choice of the algorithm. In additive mode, the algorithm reached its first stable bitrate for a short period of time. Within this period, the frame rate and video resolution also reached their maximum and stable values. This happened around 6 seconds. Finally, GCC detected that the bitrate could be increased more (see chapter 2) and switched back to multiplicative mode. A convergence time of 11 seconds is quite long for a real-time communication service. A value around 1 or 2 seconds was expected to fulfill requirement 4.

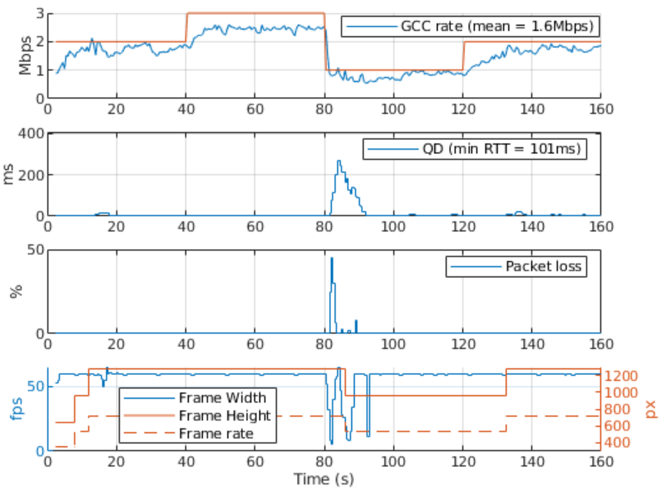
Regarding bitrate stability (requirement 5), the connection was very stable after 12 seconds. The small oscillations in bitrate correspond to the transitions between the different states of GCC, namely increase, decrease and hold. The bitrate average and standard deviation were 2.46 Mbps and 0.073 Mbps respectively. The figure also shows the good performance of the connection with respect to latency and packet loss.

6.2.2. Variable Available Capacity with a Single Flow - Experiment 2

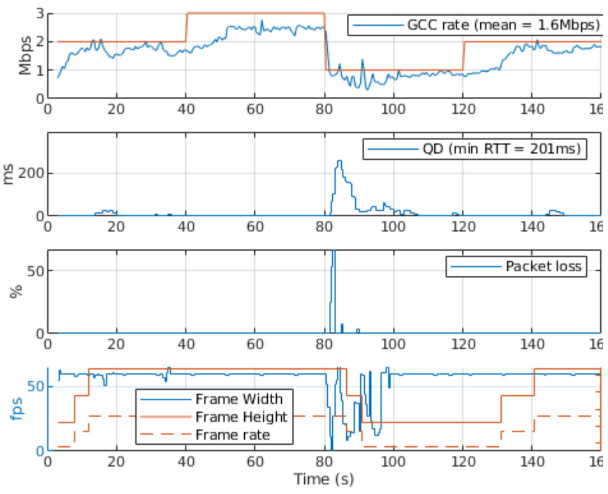
The main goal of this experiment was to determine if GCC fulfills requirements 1a and 1b. The available bandwidth of the channel was modified in order to simulate the addition or removal of a bottleneck. The experiment was divided in four sections of 40 seconds each. The sections correspond to specific values of available bandwidth. The experiment was per-



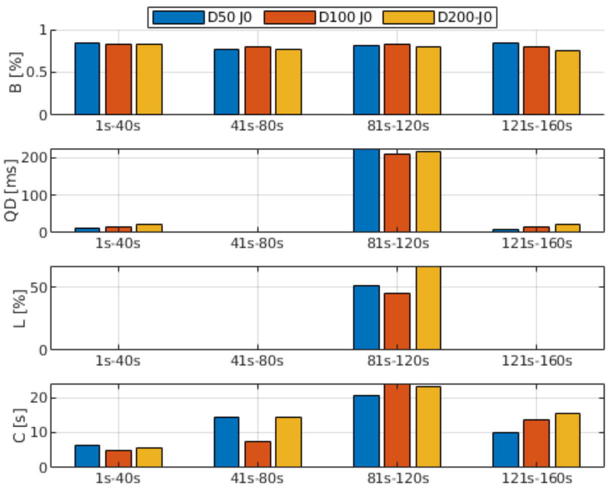
(a) RTT 50 ms



(b) RTT 100ms



(c) RTT 200ms



(d) Consolidated Results

Figure 6.2: Graphs experiment 2 - Wired network

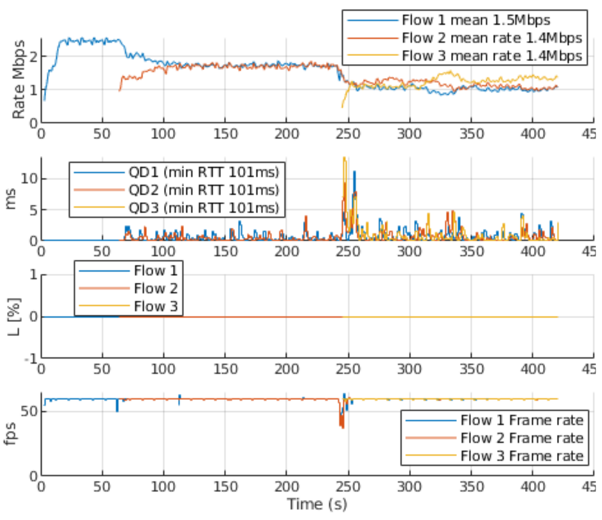
formed with different values of delays, namely 50 ms, 100 ms and 200 ms. The results are presented in figure 6.2 and table 6.1.

It is interesting to see the similarities between the results obtained with different delays. This is an expected behavior since as it was described in chapter 2, the GCC rate is mainly affected by the variation of queuing delay and not by the amount of delay in the link. Another remarkable aspect of the results, was the high bandwidth utilization. Values over 80% were observed in most cases. Section 2 is an exception since the rate is limited by the encoder. The standard deviation of bitrate was around 0.2 Mbps in all scenarios.

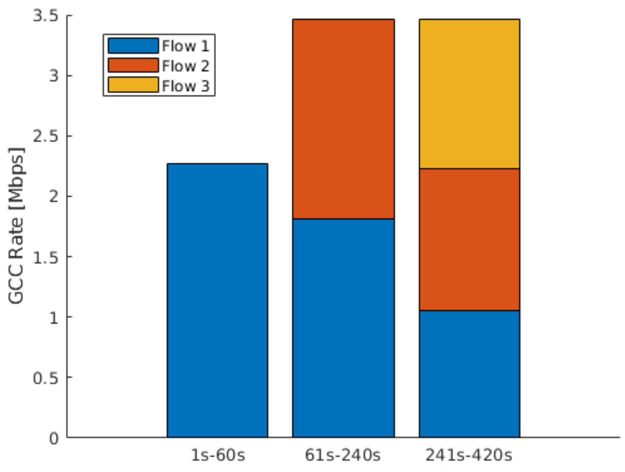
Regarding convergence time, during the first section GCC took about 5 to 6.5 seconds to reach a value close to 1.7 Mbps. This convergence time is similar to the time needed to reach the first stable bitrate (around 2 Mbps) in the previous experiment. In section 2, the maximum bitrate was achieved after approximately 14 seconds when the delay was both 50 ms and 200 ms, and 7.5 seconds when the delay was 100 ms. Since section 4 presents a converge time between 10 and 15 seconds, it can be concluded that GCC has poor conver-

Delays	Section 1	Section 2	Section 3	Section 4
50 ms				
Sending rate - mean (Mbps)	1.69	2.30	0.82	1.68
Sending rate - std (Mbps)	0.21	0.25	0.24	0.18
Channel Utilization (%)	84.5	76.7	82	84
Convergence time (s)	6.5	14.5	20.5	10
Queuing Delay - 25th perc. (ms)	0.2	0.2	0.2	0.2
Queuing Delay - 90th perc. (ms)	3.56	0.8	141.3	6.3
Queuing Delay - 95th perc. (ms)	10.84	1.7	224.5	8.2
Packet Loss - max (%)	0	0	51.2	0
100 ms				
Sending rate - mean (Mbps)	1.67	2.4	0.83	1.60
Sending rate - std (Mbps)	0.24	0.16	0.23	0.26
Channel Utilization (%)	83.5	80	83	80
Convergence time (s)	5	7.5	24	13.5
Queuing Delay - 25th (ms)	0	0	0	0.4
Queuing Delay - 90th (ms)	7.72	0	153.1	9
Queuing Delay - 95th (ms)	14.14	0.2	207.5	15.8
Packet Loss - max (%)	0	0	44.8	0
200 ms				
Sending rate - mean (Mbps)	1.64	2.3	0.80	1.51
Sending rate - std (Mbps)	0.25	0.28	0.27	0.37
Channel Utilization (%)	82	76.7	80	75.5
Convergence time (s)	5.5	14.5	23	15.5
Queuing Delay - 25th (ms)	0	0	4.6	0
Queuing Delay - 90th (ms)	17.6	0	155.9	10.1
Queuing Delay - 95th (ms)	22.85	0	216.2	22
Packet Loss - max (%)	0	0	66.9	0

Table 6.1: Results experiment 2 - Wired network



(a) RTT 100 ms



(b) Total GCC Rate

Figure 6.3: Graphs experiment 3 - Wired Network

gence when it has to increase its bitrate due to a change in available channel bandwidth. On the other hand, it is observed that GCC has a fast response when the available bandwidth is suddenly decreased. The bitrate went rapidly down to avoid an increase in delay and packet loss. However, this was not enough to avoid a peak in both, packet loss and queuing delay. The peak in packet loss reached a value above 50% and the 90th and 95th percentile of queuing delay were around 150 ms and 200 ms respectively. These peaks are difficult to avoid since they are caused by packets sent just before the change in channel bandwidth. Finally, regarding the quality of the connection during this section, the key indicator is the frame rate which was oscillating between 80 and 93 seconds. This might cause short interruptions in the communication and a decrease in video resolution.

With respect to the requirements 1a and 1b, GCC fulfills these requirements since it presented in general low queuing delay and reacted rapidly when the available channel was reduced.

6.2.3. Competing Media Flows with Same Congestion Control Algorithm - Experiment 3

This experiment mainly evaluates requirements 1a and 2 when the channel is shared with other GCC flows. The link capacity was set to 4 Mbps and the delay was set to 100 ms. The experiment was divided in 3 sections. The first section lasted 60 seconds and the duration of each of the following sections was 180 seconds. In the first section, there was no competing flows in the channel. In section 2, an additional GCC flow was added. Finally, the third section added a second competing GCC flow. The results are presented in figure 6.3 and table 6.2.

During the first 60 seconds, there was only one media flow present in the link. As expected, the sending bit rate was limited by the encoder reaching an average value of 2.27 Mbps. This corresponds to a bandwidth utilization of 56.8%. Then, for the following 180 seconds, an additional media flow was added to the link. From the figure, it can be appreciated that while the new media flow was increasing its sending bit rate, the rate of the existing flow was decreasing. The first and second flows reached an average rate equal to 1.81 Mbps and 1.66 Mbps respectively. By adding up these two values, a total GCC rate of 3.47 Mbps is obtained. This gives us a bandwidth utilization of 86.7% which confirms again the high bandwidth utilization that is achieved with GCC. If we look at how the 3.47 Mbps was shared

Flows	Section 1	Section 2	Section 3
Flow 1			
GCC rate - mean (Mbps)	2.27	1.81	1.05
GCC rate - std (Mbps)	0.42	0.17	0.11
Utilization (%)	100	52.2	30.4
Flow 2			
GCC rate - mean (Mbps)	0	1.66	1.17
GCC rate - std (Mbps)	0	0.14	0.13
Utilization (%)	0	47.8	33.9
Flow 3			
GCC rate - mean (Mbps)	0	0	1.24
GCC rate - std (Mbps)	0	0	0.12
Utilization (%)	0	0	35.7
Channel			
GCC rate (Mbps)	2.27	3.47	3.46
Utilization (%)	56.8	86.7	86.5

Table 6.2: Results experiment 3 - Wired Network

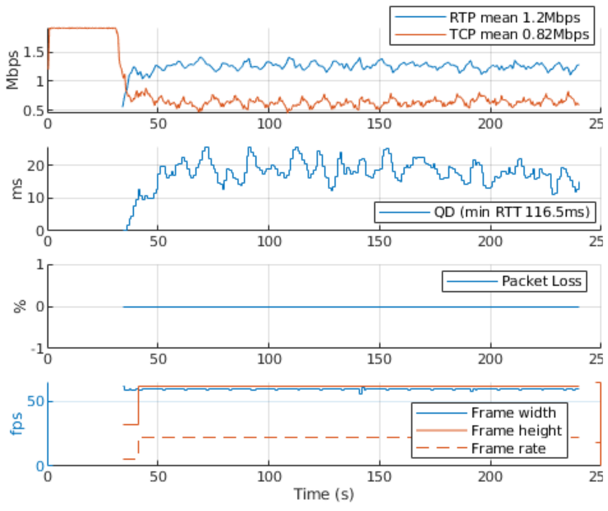
between the flows, 52.2% corresponds to flow 1 and 47.8% to flow 2. This means that GCC in both flows was able to fairly share the resources of the channel. Additionally, there was no extra queuing delay and the packet loss was equal to 0%.

Finally, when a third media flow was added to the link for the last 180 seconds, it is observed that there was a slight increase in queuing delay. However, this peak was short and only reached a value around 10 ms. We can also see that GCC managed to keep the packet loss equal to 0 %. By looking at the frame rate when this third flow was added to the channel, this value for all flows was unstable for 10 seconds. However, it did not reach values below 40 fps meaning that users were not affected by this disturbance. It is interesting to observe that GCC maintained the bandwidth utilization and decreased the sending bit rate of the two existing links in order to equally divide the resources of the link. This time, flows 1, 2 and 3 present a rate utilization equal to 30.4 %, 33.9 % and 35.7 % respectively. It is concluded that GCC is a fair algorithm when it has to share resources with other real-time flows that also implement GCC as their congestion control algorithm. Furthermore, it is able to achieve this while having a stable sending bit rate. The standard deviation of the flows was around 0.1 Mbps. With regards to this experiment, GCC fulfilled requirements 1a and 2.

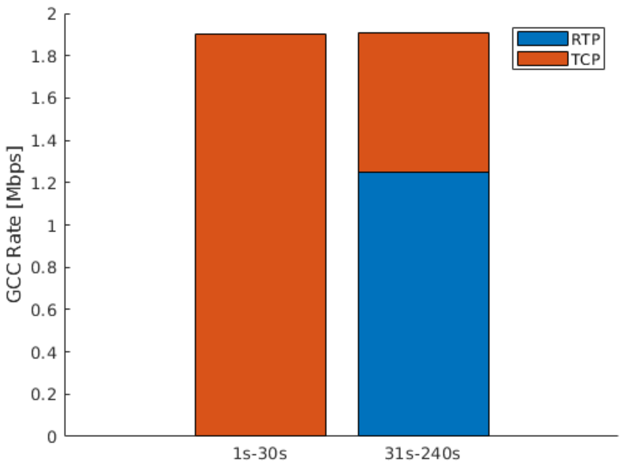
6.2.4. Media Flow Competing with a Long-lived TCP Flow - Experiment 4

The objective of this experiment was to evaluate if GCC fulfills requirements 1c, 2 and 3. The channel bandwidth was set to 2 Mbps and the delay was set to 100 ms. Since it was expected that the long-lived TCP flow filled up the queue buffer, the queue sizes 300 ms and 1000 ms were used for comparison purposes. The experiment was divided in two sections. The duration of the first and second sections was 30 and 210 seconds respectively. In section 1, there was no competing flows in the channel. Section 2 added a competing TCP flow. The results are presented in figure 6.4 and table 6.3.

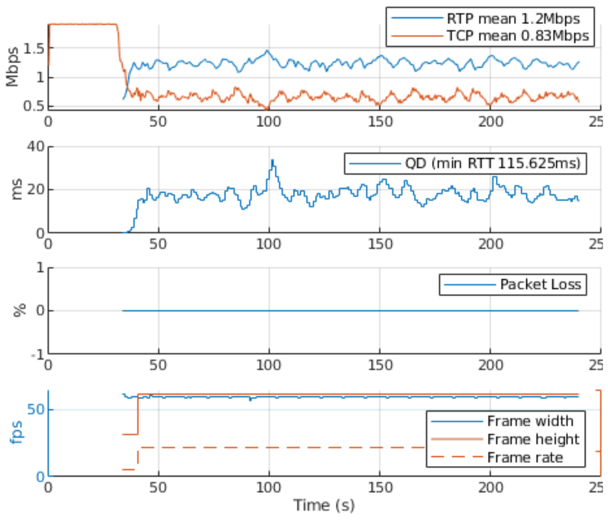
Initially, this experiment was performed with a queue size equal to 300 ms. During the first 30 seconds of the experiment, only the TCP flow was present in the link. This flow reached an average rate equal to 1.9 Mbps. Then, once the media flow was introduced in the link, the rate of TCP rapidly decreased. It reached an average value equal to 0.66 Mbps. On the other hand, the rate of the media flow was increasing in multiplicative mode and reached an average value equal to 1.25 Mbps. This media flow represents 64.5% of the total measured rate. Additionally, it can be observed that there was no packet loss and the frame



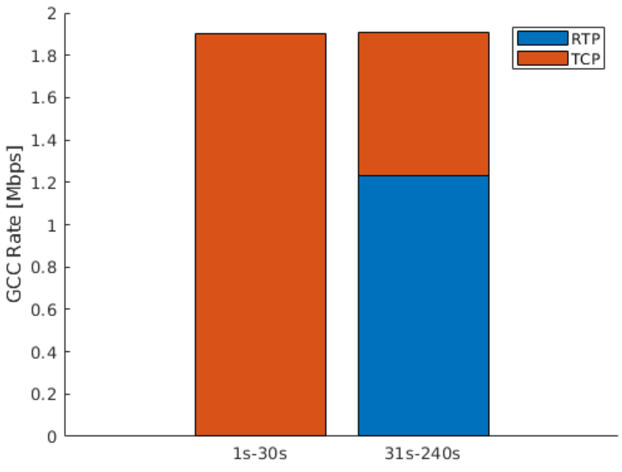
(a) Queue Size 300 ms - RTT 100 ms



(b) Total GCC Rate



(c) Queue Size 1000 ms - RTT 100 ms



(d) Total GCC Rate

Figure 6.4: Graphs experiment 4 - Wired Network

Flows	Queue Size 300 ms		Queue Size 1000 ms	
	Section 1	Section 2	Section 1	Section 2
GCC				
Rate - mean (Mbps)	0	1.25	0	1.23
Rate - std (Mbps)	0	0.09	0	0.09
Queuing delay - mean (ms)	0	19.67	0	18.36
RTT - min (ms)	0	116.5	0	115.63
Utilization (%)	100	65.4	100	64.5
TCP				
Rate - mean (Mbps)	1.9	0.66	1.9	0.68
Rate - std (Mbps)	0.09	0.15	0.09	0.15
Utilization (%)	0	34.6	0	35.5
Channel				
Rate (Mbps)	1.9	1.91	1.9	1.91
Utilization (%)	95	95.5	95	95.5

Table 6.3: Results experiment 4 - Wired Network

rate was kept stable around 60 fps. This confirms the good quality of the video communication and the capacity of GCC to not collapse when a loss-based flow (TCP) is also present in the link. Regarding the queuing delay, the TCP flow influenced this parameter, but the increase and variation of it did not affect the quality of the communication. The minimum delay was increased from 101 ms to 115.63 ms compared to previous experiments, and the mean queuing delay was only 19.67 ms. Similar results were obtained with a queue size equal to 1000 ms.

In conclusion, GCC managed to keep a low queuing delay despite the presence of a TCP flow. Moreover, it did not starve or was starved by this loss-based congestion control flow. The sharing of the channel bandwidth went according to the requirements as the bandwidth should not be equally divided since the flow types are different. GCC leaves enough bandwidth to TCP to continue performing well.

6.3. WiFi Network

The following subsections present the results of the experiments in WiFi. The main objective is to evaluate the performance of a WebRTC call when other GCC flows and TCP flows are competing for resources. This analysis focuses on the performance of an individual WebRTC call (GCC flow and video quality). GCC flows used to congest the channel are not analyzed since their performance was limited by the CPU power of the laptops that handled these connections. The first two experiments evaluate the uplink and downlink connections separately since WiFi works in half-duplex mode (uplink and downlink connections compete for resources). Experiments 3 and 4 use a bidirectional (uplink and downlink) WebRTC call. Experiments evaluate 3 scenarios which were described in chapter 5.3. This section also presents the signal strength and theorical bit rate (from Modulation Coding Scheme MCS code) of the wireless connection for each experiment. This information indicates the dynamic conditions of the wireless channel caused mainly by multipath fading and shawdowing.

6.3.1. Distributed Coordination Function (DCF) - IEEE 802.11

In order to understand the results of the experiments (especially the last two), it is necessary to explain the basics of the Distributed Coordination Function (DCF) protocol. This protocol is the fundamental and most used random access technique for Medium Access Control (MAC) in IEEE 802.11 networks. DCF operates with Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with binary exponential backoff algorithm. In a nutshell, a node needs to sense the channel before starting a transmission in order to detect if another node is transmitting. The transmission begins if the channel is free for a DCF interframe space

Scenarios	Section 1	Section 2	Section 3
Scenario 1	Uplink		
GCC rate - mean (Mbps)	2.32	2.39	2.41
GCC rate - std (Mbps)	0.34	0.12	0.1
Delay - mean (ms)	4.13	3.65	6.49
Delay - 25th perc. (ms)	2.19	2.25	3
Delay - 50th perc. (ms)	3.25	2.88	4.63
Delay - 90th perc. (ms)	8.55	7.13	12.88
Delay - 95th perc. (ms)	9.99	8.5	18
Packet Loss - mean (%)	0.13	0.03	0.03
Packet Loss - max (%)	0.35	0.18	0.17
Scenario 2	Uplink		
GCC rate - mean (Mbps)	1.81	1.35	0.85
GCC rate - std (Mbps)	0.24	0.32	0.13
Delay - mean (ms)	7.97	8.76	8.93
Delay - 25th perc. (ms)	2.94	3.5	5.38
Delay - 50th perc. (ms)	4.25	7.13	8.88
Delay - 90th perc. (ms)	23.10	19.50	15.63
Delay - 95th perc. (ms)	24.40	25.50	17.75
Packet Loss - mean (%)	0.12	0.02	0.06
Packet Loss - max (%)	2.85	0.22	0.42

Table 6.4: Results uplink experiment 1 - WiFi Network

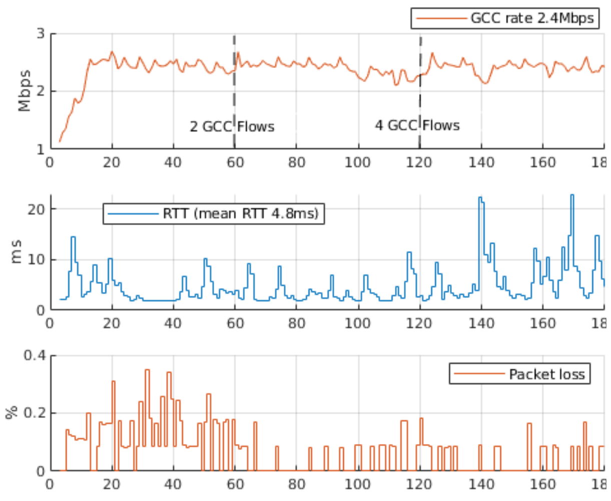
(DIFS) time interval. On the other hand, if the channel is busy, the node waits for a DIFS time interval after the transmission has ended, and then selects a random backoff value from a contention window (CW) range. The random backoff delay decrements only if the channel is free. The node starts its transmission when this delay reaches zero and the channel is not occupied. DCS does not prioritize traffic and gives equal opportunity to access the channel to all nodes in the network.

There are two otherrandom access techniques for IEEE 802.11 networks. These are called Point Coordination Function (PCF) and Hybrid Coordination Function (HCF). PCF uses the AP (Access Point) to coordinate the communication within the network. On the other hand, HCF enhances DCF and PCF. It defines traffic categories to assign different transmission priorities to frames and enables the transmission of multiple frames within a time interval.

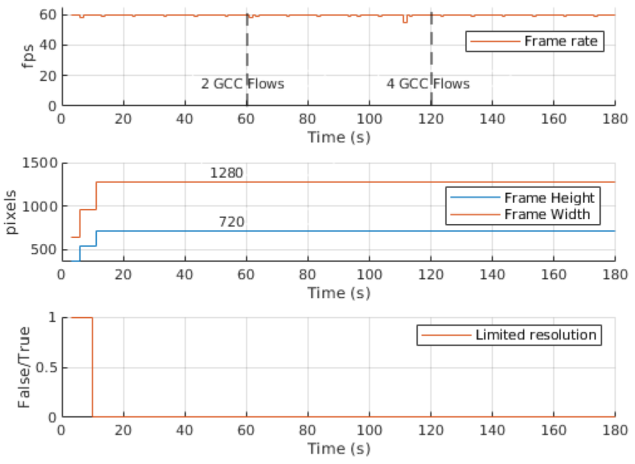
6.3.2. Competing Media Flows Sharing the Wireless Uplink - Experiment 1

The main objective of this experiment was to evaluate the quality of an uplink WebRTC call (unidirectional video call from laptop to AP) when GCC traffic was also present in the wireless channel. The experiment was divided in 3 sections of 60 seconds each. In the first section, there was no competing traffic in the channel. Two uplink GCC flows were added in section 2. Finally, in section 3, 4 competing uplink GCC flows were present in the channel. The evaluation of GCC is based on requirements 1a, 1b, 2 (partial) and 5. The results are presented in figure 6.5 and table 6.4.

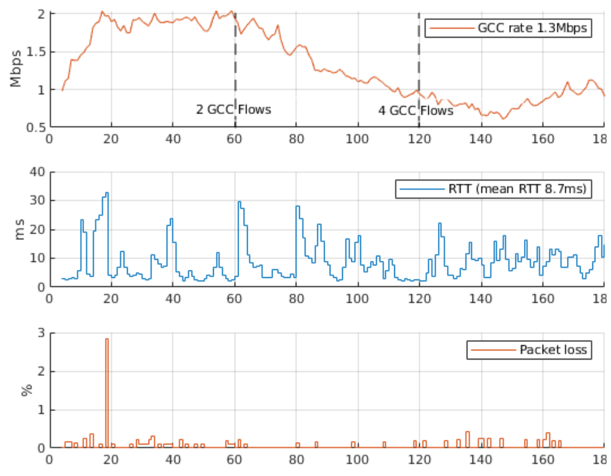
In scenario 1, where the laptops were located close to the AP, the bitrate of the GCC flow was not affected by the presence of competing GCC traffic. As observed in figure 6.5a, the bitrate did not decrease when adding 2 competing GCC flows at 60 seconds neither when 4 competing GCC flows were present in the channel after 120 seconds. This confirms the good quality of the channel (mean signal strength -41.96 dBm and mean Tx MCS bitrate 130 Mbps) since the additional GCC traffic was not able to congest the network. The average bitrate during the whole experiment was 2.4 Mbps. GCC managed to keep a low bitrate variation, low delay and packet loss almost equal to zero. With regards to video quality, the



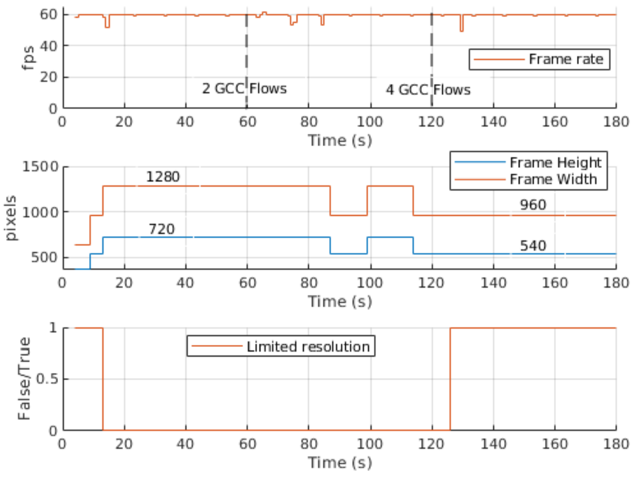
(a) Flow - Scenario 1



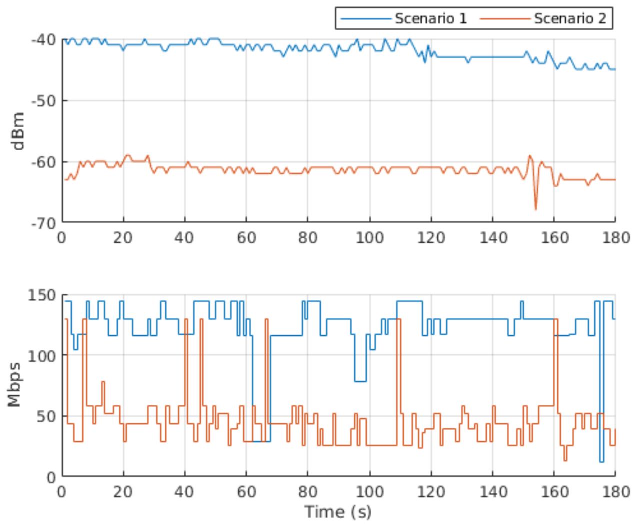
(b) Video - Scenario 1



(c) Flow - Scenario 2



(d) Video - Scenario 2



(e) Signal strength & MCS bitrate

Figure 6.5: Graphs uplink experiment 1 - WiFi Network

frame rate was kept stable around 60 fps and the resolution of the video was HD (1280x720 pixels). There was no indication of limited resolution due to bandwidth from WebRTC. In overall, the results are similar to the ones obtained in an unconstrained wired network.

A different situation occurred in scenario 2, where laptops were located in a room not close to the AP and with obstacles in between laptops and AP. The quality of the channel decreased in this scenario as presented in figure 6.5e. The average signal strength was -61.43 dBm and the mean theorical MCS bitrate was 43.4 Mbps. Under these network conditions, the channel was congested when additional GCC traffic was added. Between 60 and 120 seconds, when 2 additional GCC flows were present in the channel, the bitrate went from 2.4 Mbps to around 1 Mbps. At 120 seconds, 2 additional GCC flows were added and the bitrate continued decreasing until reaching a minimum of about 0.65 Mbps. Then, the GCC state machine went back to the increase state to settle around 1 Mbps. Once again, GCC managed to maintain a low delay (mean 8.93 ms). However, its 90th and 95th percentiles revealed that there was more variation compared to scenario 1. The packet loss increased, but it was still close to zero. It was expected to see a considerable deterioration of the video quality due to the reduction in bitrate. However, as observed in figure 6.5d, the frame rate was stable around 60 fps and the resolution of the video was set 960x540 pixels which is still higher than SD (Standard Definition - 720x480 pixels). This time, WebRTC reported limited resolution due to bandwidth.

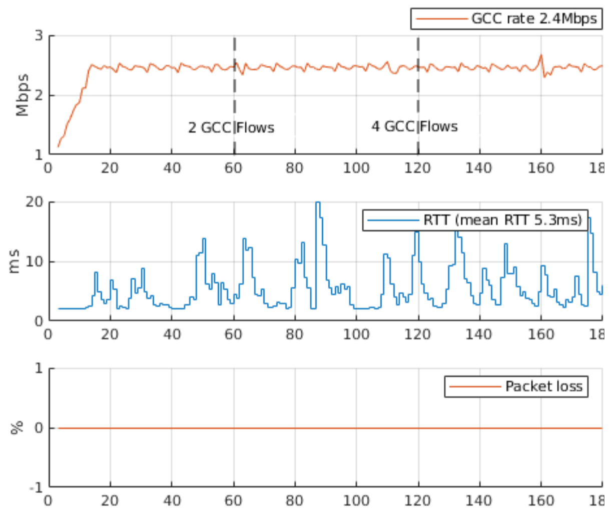
This experiment confirmed the ability of GCC to adapt its bitrate during an uplink transmission when it is competing with other GCC flows. It reacted quickly to network congestion to keep the delay as low as possible. This behavior fulfills requirements 1a and 1b. It could also be appreciated that the bitrate of GCC was reduced to share the channel with other GCC flows. Then, it switched back to the increase state to reach a fair value (requirement 2). Finally, the low standard deviation of bitrate shows once again the good stability of GCC.

6.3.3. Competing Media Flows Sharing the Wireless Downlink - Experiment 2

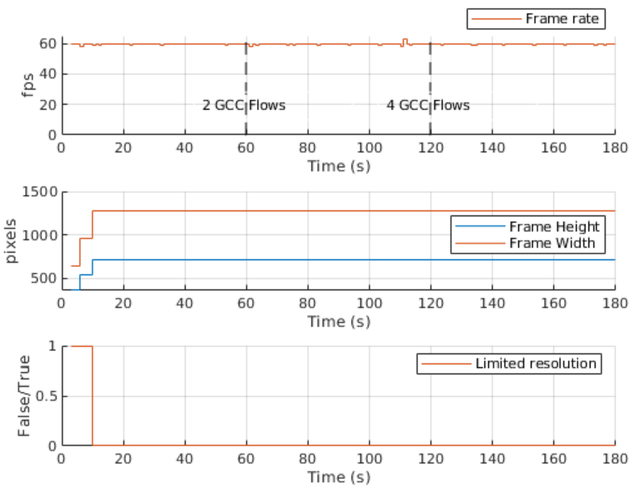
The aim of this experiment was to evaluate the quality of a downlink WebRTC call (unidirectional video call from AP to laptop) when GCC traffic was also present in the channel. The experiment was divided in 3 sections of 60 seconds each. In the first section, there was no competing traffic in the channel. Two downlink GCC flows were added in section 2. In section 3, 4 competing downlink GCC flows were present in the channel. The evaluation of GCC is based on requirements 1a, 1b, 2 (partial) and 5. The results are presented in figure 6.6 and table 6.5.

In scenario 1, as expected, the performance of GCC was better than the one obtained in the uplink direction. The bitrate was a bit higher and more stable. This can be observed in figure 6.6a. The 90th percentile of delay was under 13 ms. The packet loss was equal to zero. Regarding the video quality, the frame rate was stable around 60 fps and the video had HD resolution. WebRTC did not detect limitation of resolution due to bandwidth. The average signal strength was -42.09 dBm and the mean MCS bitrate was 144.4 Mbps.

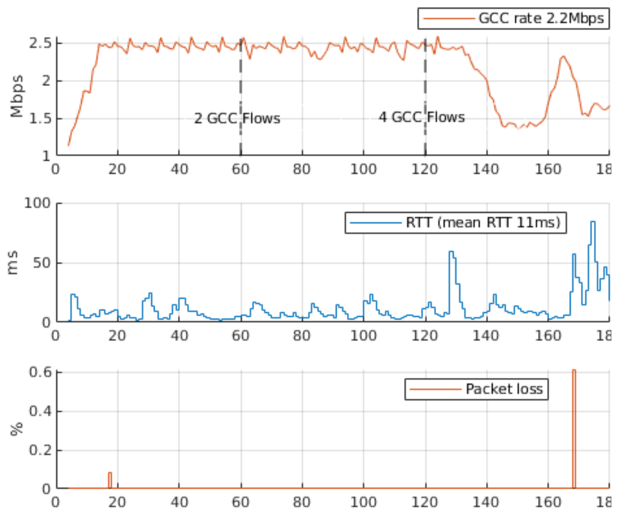
The results in scenario 2 show a different behavior than the uplink results. First, the addition of 2 flows at 60 seconds did not affect the performance of the WebRTC call. This means that the channel was able to handle 3 downlink GCC flows without being congested. The mean signal strength and MCS bitrate were -65.47 dBm and 78 Mbps respectively. At 120 seconds, 2 more GCC flows were added to the channel. This time, the downlink WebRTC call detected the presence of these flows and reduced its bitrate to approximately 1.5 Mbps in order to maintain a low delay and no packet losses. GCC was momentarily stable at this rate. At 160 seconds, GCC incremented its bitrate as a reaction of low queuing delay. This situation slightly affected the quality of the call by increasing the delay (90th and 95th percentiles of delay equal to 48.38 and 58.5 ms respectively) and generating drops in the frame rate. Nevertheless, GCC reacted fast and decreased again the bitrate to restore the previous



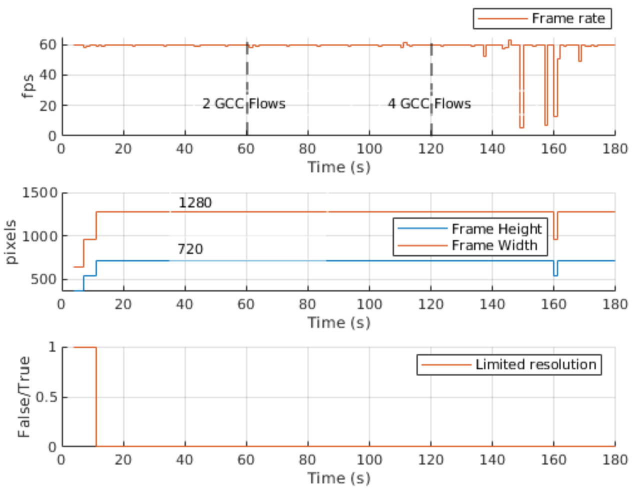
(a) Flow - Scenario 1



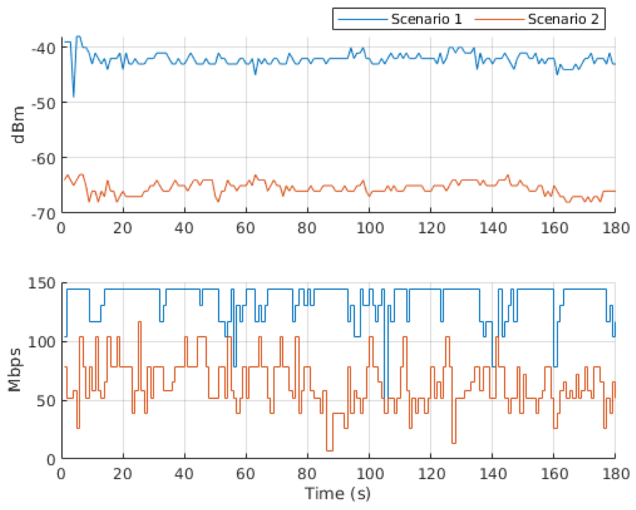
(b) Video - Scenario 1



(c) Flow - Scenario 2



(d) Video - Scenario 2



(e) Signal strength & MCS bitrate

Figure 6.6: Graphs downlink experiment 2 - WiFi Network

Scenarios	Section 1	Section 2	Section 3
Scenario 1	Downlink		
GCC rate - mean (Mbps)	2.32	2.46	2.45
GCC rate - std (Mbps)	0.34	0.05	0.05
Delay - mean (ms)	4.07	5.64	6.13
Delay - 25th perc. (ms)	2	2.38	3.25
Delay - 50th perc. (ms)	3	4.5	4.88
Delay - 90th perc. (ms)	7.8	12	12.25
Delay - 95th perc. (ms)	10.21	13.5	14.88
Packet Loss - mean (%)	0	0	0
Packet Loss - max (%)	0	0	0
Scenario 2	Downlink		
GCC rate - mean (Mbps)	2.33	2.44	1.89
GCC rate - std (Mbps)	0.33	0.07	0.04
Delay - mean (ms)	7.71	7.6	17.28
Delay - 25th perc. (ms)	2.75	4.38	6.13
Delay - 50th perc. (ms)	5.75	5.75	9.13
Delay - 90th perc. (ms)	19.65	15.38	48.38
Delay - 95th perc. (ms)	20.74	17.38	58.5
Packet Loss - mean (%)	0.001	0	0.01
Packet Loss - max (%)	0.08	0	0.62

Table 6.5: Results downlink experiment 2 - WiFi Network

performance. Opposite to the previous experiment, WebRTC did not indicate low resolution due to bandwidth.

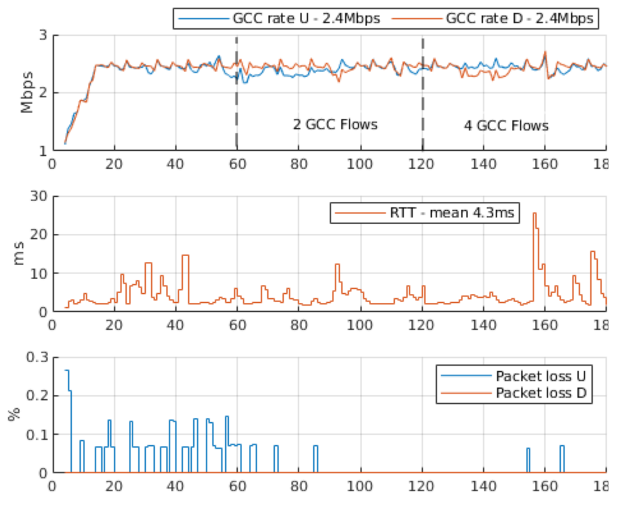
Similar to the performance obtained in the uplink direction, GCC was able to keep a low delay while competing with other GCC flows in the downlink direction. The performance was better than the uplink direction due to less contention in the channel and hardware of the AP. It is concluded that GCC fulfilled requirements 1a, 1b, 2 and 5.

6.3.4. Competing Media Flows in a Bidirectional Transmission - Experiment 3

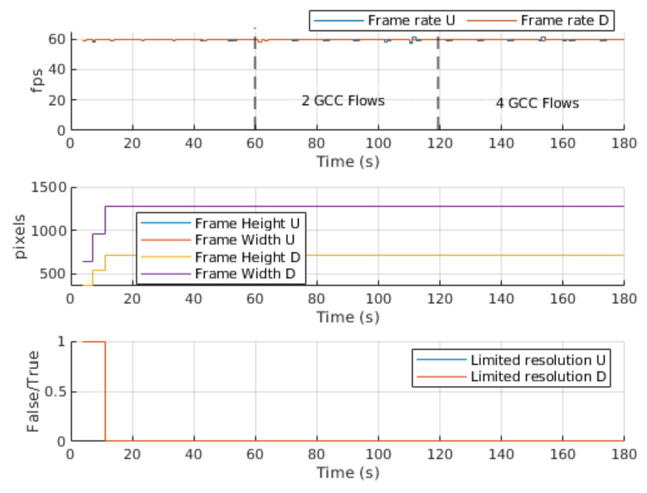
This experiment evaluates the performance of a bidirectional WebRTC call (uplink and downlink video calls) when there are other GCC flows competing for resources in the channel. The experiment was divided in 3 sections of 60 seconds. The first section did not have competing traffic in the channel. In section 2, two uplink and two downlink GCC flows were added. Section 3 had four uplink and four downlink GCC flows as competing traffic. The evaluation of GCC is based on requirements 1a, 1b, 2 (partial) and 5. The results are presented in figure 6.7 and table 6.6.

The results of scenario 1 are presented in figures 6.7a and 6.7b. The bitrate of both, uplink and downlink GCC flows was not affected by the competing traffic. The conditions of the channel were good and this was reflected with a high and stable bitrate. The average value was 2.4 Mbps and the standard deviation in the second and third sections was below 0.1 Mbps. Regarding the delay, the mean value was under 5 ms and 95 percent of the measurements were below 15 ms. The packet loss was near zero. With respect to the video quality, the resolution was HD and the frame rate was stable around 60 fps. In conclusion, the performance of the WebRTC call is similar to the one obtained in an unconstrained wired network.

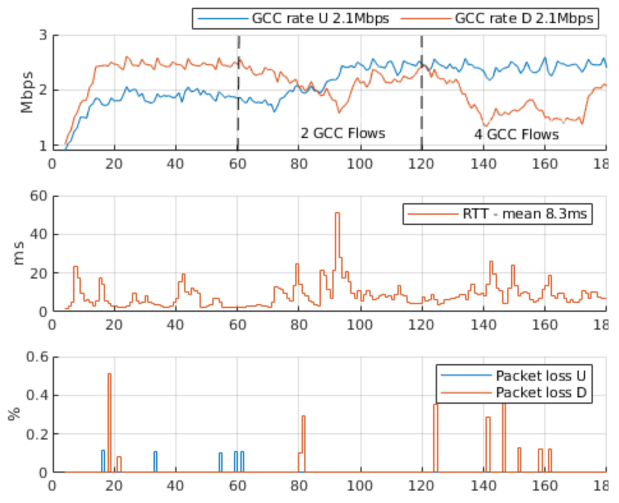
In scenario 2, the quality of the connection decreased. The average signal strength was -63.03 dBm. The network conditions for the 3 scenarios are presented in figure 6.8. The GCC performance in scenario 2 is presented in figure 6.7c. During the first 60 seconds of the experiment there was no competing traffic in the channel. However, since WiFi operates in



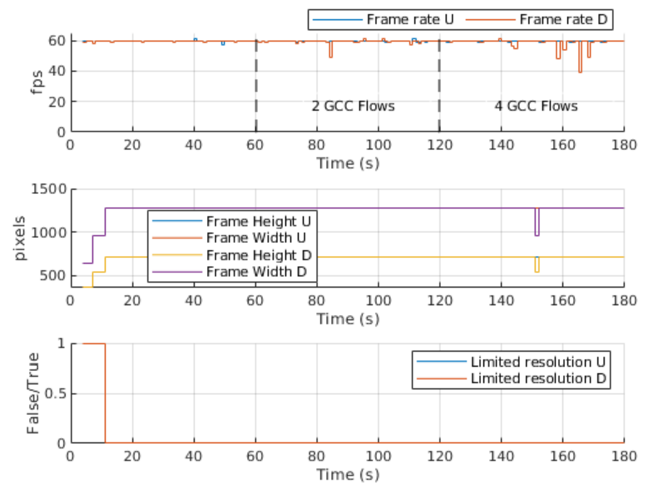
(a) Flow - Scenario 1



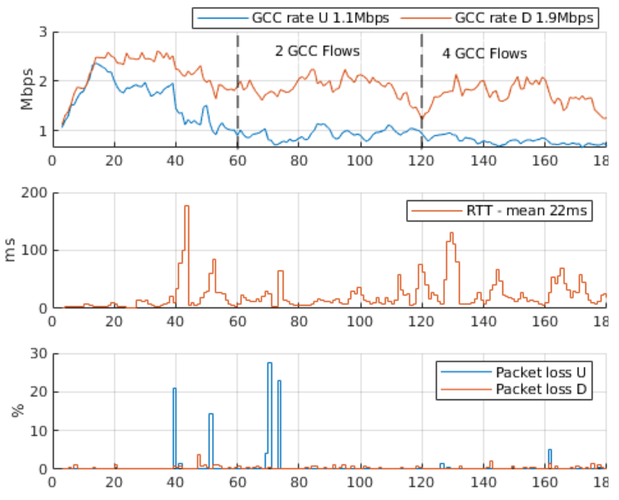
(b) Video - Scenario 1



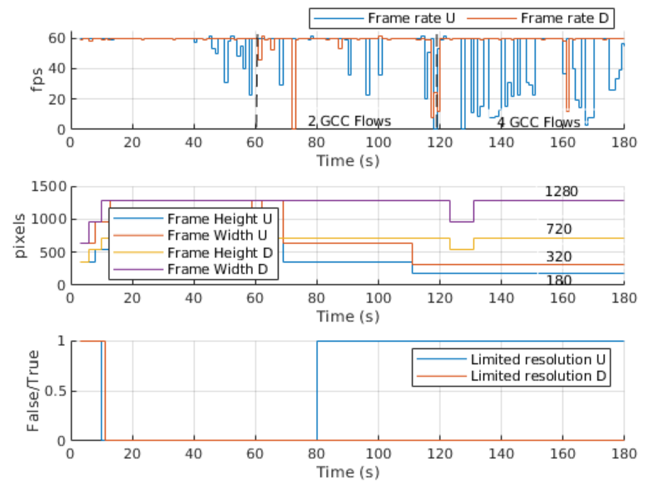
(c) Flow - Scenario 2



(d) Video - Scenario 2



(e) Flow - Scenario 3



(f) Video - Scenario 3

Figure 6.7: Graphs bidirectional experiment 3 - WiFi Network (U = Uplink and D = Downlink)

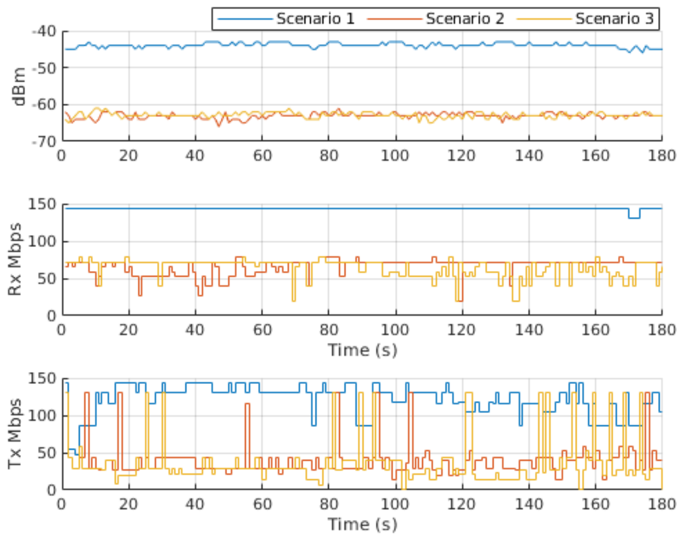


Figure 6.8: Signal strength & MCS bitrate experiment 3 - WiFi Network

Scenarios	Section 1	Section 2	Section 3	Section 1	Section 2	Section 3
Scenario 1		Uplink			Downlink	
GCC rate - mean (Mbps)	2.31	2.39	2.44	2.33	2.44	2.42
GCC rate - std (Mbps)	0.31	0.09	0.07	0.33	0.08	0.1
Delay - mean (ms)	4.28	3.63	4.93	4.28	3.63	4.93
Delay - 25th perc. (ms)	2.35	2.2	2.6	2.35	2.2	2.6
Delay - 50th perc. (ms)	3	3	3.4	3	3	3.4
Delay - 90th perc. (ms)	9.16	6	10.3	9.16	6	10.3
Delay - 95th perc. (ms)	12.8	6.7	14.6	12.8	6.7	14.6
Packet Loss - mean (%)	0.05	0	0	0	0	0
Packet Loss - max (%)	0.26	0.07	0.07	0	0	0
Scenario 2		Uplink			Downlink	
GCC rate - mean (Mbps)	1.80	2.16	2.41	2.31	2.16	1.77
GCC rate - std (Mbps)	0.25	0.29	0.87	0.36	0.2	0.3
Delay - mean (ms)	6.09	9.48	9.09	6.09	9.48	9.09
Delay - 25th perc. (ms)	2.69	4.13	6.13	2.69	4.13	6.13
Delay - 50th perc. (ms)	4.75	7.88	8.5	4.75	7.88	8.5
Delay - 90th perc. (ms)	13.7	19.88	13.88	13.7	19.88	13.88
Delay - 95th perc. (ms)	17.58	23.13	18.88	17.58	23.13	18.88
Packet Loss - mean (%)	0.01	0	0	0.01	0.01	0.02
Packet Loss - max (%)	0.11	0.11	0	0.51	0.29	0.37
Scenario 3		Uplink			Downlink	
GCC rate - mean (Mbps)	1.6	0.93	0.79	2.05	1.97	1.65
GCC rate - std (Mbps)	0.42	0.11	0.07	0.35	0.2	0.25
Delay - mean (ms)	17	16.51	31.48	17	16.51	31.48
Delay - 25th perc. (ms)	2.75	8	11.38	2.75	8	11.38
Delay - 50th perc. (ms)	7.5	12.63	21.63	7.5	12.63	21.63
Delay - 90th perc. (ms)	32.35	31.38	68.38	32.35	31.38	68.38
Delay - 95th perc. (ms)	82.08	47.38	96.25	82.08	47.38	96.25
Packet Loss - mean (%)	0.67	0.92	0.15	0.31	0.24	0.26
Packet Loss - max (%)	21	27.47	5.08	3.69	1	2.06

Table 6.6: Results bidirectional experiment 3 - WiFi Network

half-duplex mode, the GCC flows in these two directions are competing between each other. It can be observed that in this section, the downlink bitrate was higher. Its mean value was 2.31 Mbps whereas the mean uplink bitrate was 1.80 Mbps. The mean delay was equal to 6.09 ms.

At 60 seconds, two additional GCC connections were added to the channel. The downlink bitrate started decreasing as a response to the increase in delay caused by these new downlink traffic. The AP had more frames to transmit but the same amount of channel resources due to DCF. At 92 seconds, the bitrate reached a minimum value around 1.8 Mbps. At this point, the bitrate started increasing again because the queuing delay was reduced. It then reached a value around 2.2 Mbps at 120 seconds. On the other hand, the uplink bitrate increased its value during this section. It also reached a value around 2.2 Mbps at 120 seconds. From the figure, it is possible to observe an increase in delay. Therefore, it was expected a decrease in bitrate for both, uplink and downlink connections. However, GCC reacts to variation in queuing delay and not to end-to-end latency. In this case, the delay (end-to-end latency) increased due to the downlink connection. The queuing delay in the uplink connection increased as a consequence of the reduction in downlink bitrate. The data from the table shows that the 90th percentile of delay in this section was below 24 ms.

At 120 seconds, again two additional GCC flows were added. A similar situation occurred in this section. The downlink bitrate decreased because of the new GCC traffic and the queuing delay in the uplink connection did not change. The average uplink and downlink bitrates were 2.41 and 1.77 Mbps respectively. The packet loss during this experiment was very close to zero. Regarding the video quality, it was expected to have some degradation due to the variations in bitrate but as it can appreciated in figure 6.7d, the frame rate was stable and the resolution was HD during the whole experiment.

The configuration of scenario 3 places the laptop that handles the competing GCC connections close to the AP, whereas the evaluated GCC is in the same location as scenario 2. The results are presented in figure 6.7e. The uplink and downlink GCC flows competed only with each other during the first 60 seconds. It was expected to have a similar behavior as the one observed in scenario 2, but due to the changing conditions of the wireless channel, the bitrates, especially the uplink traffic, behaved differently. The uplink traffic presented a maximum peak of packet loss equal to 21%, which as described in chapter 2, reduced rapidly the bitrate. On the other hand, there was an increase in delay for the downlink connection which caused a reduction in bitrate.

At exactly 60 seconds, the uplink and downlink bitrates were about 1 Mbps and 2 Mbps respectively. Also at this time, two additional bidirectional GCC flows were added to the channel. During this section, the uplink bitrate was decreased further due to peaks in packet loss (max. equal to 27.47%) and the presence of competing traffic. The mean uplink bitrate in this section was 0.93 Mbps. With respect to the downlink flow, this had a good performance. Its bitrate was around 2 Mbps. In this section, the 95th percentile of delay was 23.13 ms.

At 120 seconds, two additional bidirectional GCC flows were added. The uplink bitrate decreased further presenting a mean value equal to 0.79 Mbps. The downlink bitrate was also reduced due to an increase in queuing delay. The mean value was equal to 1.65 Mbps. With respect to the video quality, the downlink transmission managed to keep a HD resolution and a frame rate stable with just a few short drops. The uplink transmission had issues before and after adding the competing traffic. At the end of the experiment the resolution was 320x180 pixels which is lower than SD resolution. The WebRTC API reported limited resolution due to bandwidth at 80 seconds and after 120 seconds, the frame rate was very unstable meaning that the communication was very poor.

In conclusion, GCC reacted quickly to maintain a low delay as it was appreciated in the results of scenario 1 and 2. It was also fair with other GCC flows. Although, the uplink and downlink flows did not equally share the resources when they were the only ones competing. The bitrate was not stable, however, the video quality was not affected by this. The main issue that limited the performance of WebRTC in scenario 3 was the well-known near far problem in wireless communication.

6.3.5. Media Flow Competing with a Long-lived TCP Flow - Experiment 4

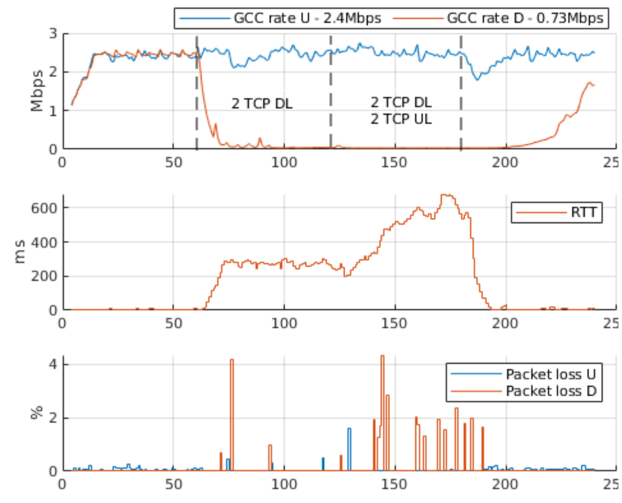
The aim of this experiment was to evaluate the performance of a WebRTC call when long-lived TCP flows were present in the channel. The experiment was divided in 4 sections of 60 seconds each. In the first section, there was no competing traffic. Then, in section 2, two TCP flows were transmitting from the AP towards the laptops (downlink). After that, in section 3, two TCP flows transmitting from the laptops towards the AP were added (uplink). Finally, the last section did not have again competing traffic. The evaluation of GCC is based on requirements 1, 2, 3 and 5. The results are presented in figure 6.9 and table 6.7. The signal strength and MSC bitrate of all 3 evaluated scenarios are presented in figure 6.10.

The results obtained in scenario 1 are presented in figure 6.9a. The first section shows that the bitrate of both uplink and downlink was high and reasonably stable. The mean rates of the uplink and downlink connections were 2.29 Mbps and 2.32 Mbps respectively. Then, as soon as the two downlink TCP flows were added to the channel, the downlink bitrate of GCC collapsed. The mean values of downlink bitrate and delay were 0.24 Mbps and 231.63 ms respectively. The delay increased even further in the third section reaching a mean of 445.44 ms. The mean downlink bitrate for this section was 0.04 Mbps. On the other hand, the uplink bitrate of GCC remained high and stable during these two sections as observed in table 6.7. The queuing delay (this is not end-to-end delay) in the uplink direction was not affected by the addition of the competing traffic. The collapse of downlink bitrate was due to the DCF protocol. The AP had more traffic to transmit (2 additional TCP flows each one coming from a different device) but the same allocation of channel resources. One can argue that the laptops also had more traffic to transmit when the two uplink TCP flows were activated. However, this traffic was not aggregated in a single device as it happened with the AP. The TCP rates are presented in figure 6.11.

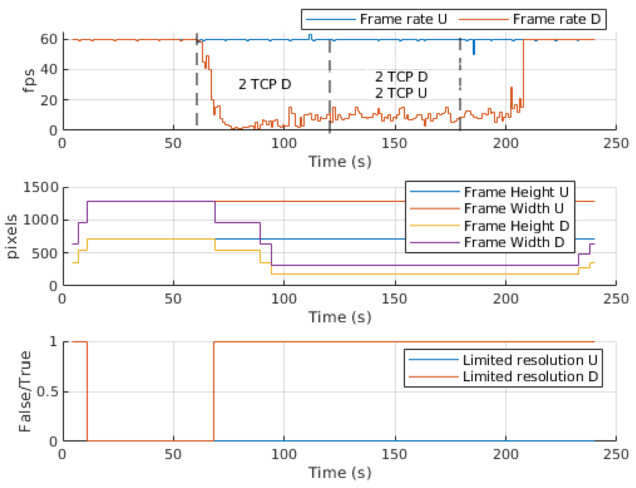
With respect to the video quality, it can be appreciated from figure 6.9b, that the frame rate of the downlink connection rapidly decreased after adding the TCP flows. For sections 2 and 3, the frame rate was below 10 fps. The resolution also decreased and reached 320x180 pixels. In conclusion, it was not possible to display the downlink video call. The uplink frame rate and resolution remained high and stable during these two sections. Finally, the fourth section shows once again the slow convergence of GCC. It took about 20 seconds after disconnecting the TCP flows for GCC to slowly start increasing its bitrate.

Figures 6.9c and 6.9d present the results obtained in scenario 2. The behavior of the downlink bitrate is similar to the one observed in scenario 1. However, the delay was higher due to the channel conditions of this scenario. The mean delay in sections 2 and 3 was equal to 381.9 ms and 622.23 ms respectively. The packet loss on the downlink connection also increased. The maximum value was 9.76%. The uplink bitrate was lower than in the previous scenario but this was mainly due to the degradation of the channel conditions (figure 6.10). The mean bitrate went from 1.76 Mbps in the first section to 1.34 Mbps in the second and third sections. This means that this time, the TCP flows affected the queuing delay in the uplink connection. However, by looking at the results of the video quality, the frame rate and resolution were not impacted by this cross traffic. The only exception happened around 150 seconds where the delay reached a value above 1 second. The resolution decreased one level (1080x560 pixels), but it was still between SD and HD

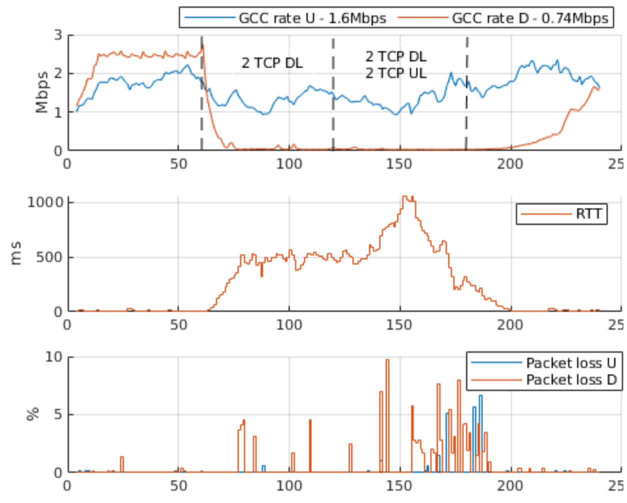
Finally, the results of scenario 3 are presented in figures 6.9e and 6.9f. The downlink bitrate and video quality again collapsed when the TCP flows were added to the channel. Opposite to the previous scenarios, the uplink connection also collapsed with the presence of the TCP flows. Since the cross traffic was being generated close to the AP, the laptop



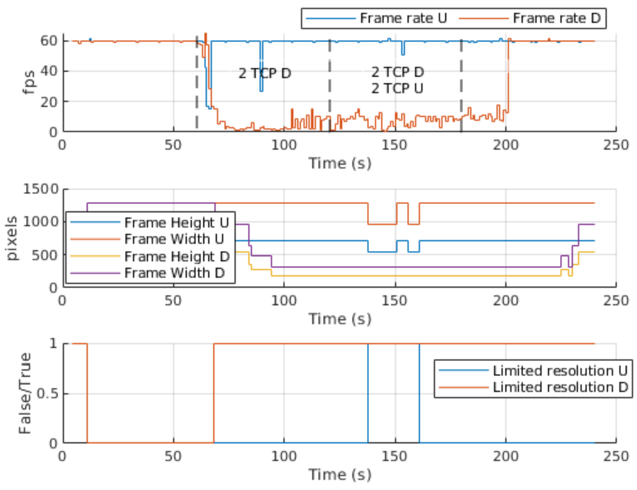
(a) Flow - Scenario 1



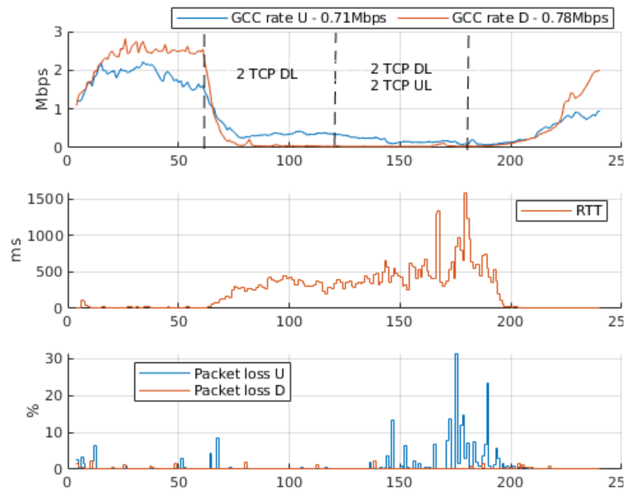
(b) Video - Scenario 1



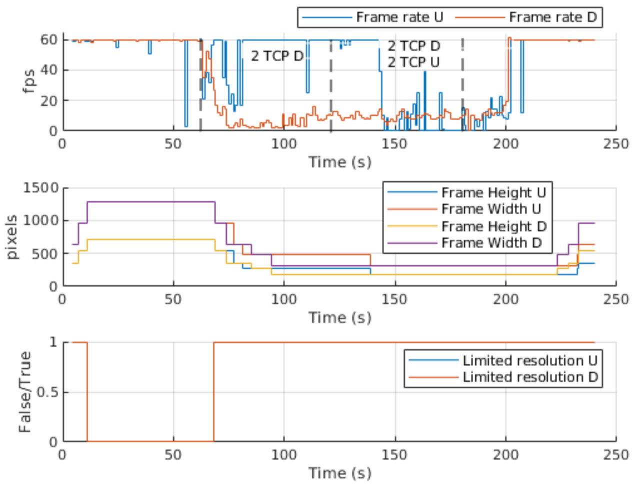
(c) Flow - Scenario 2



(d) Video - Scenario 2



(e) Flow - Scenario 3



(f) Video - Scenario 3

Figure 6.9: Graphs TCP cross traffic experiment 4 - WiFi Network (U = Uplink and D = Downlink)

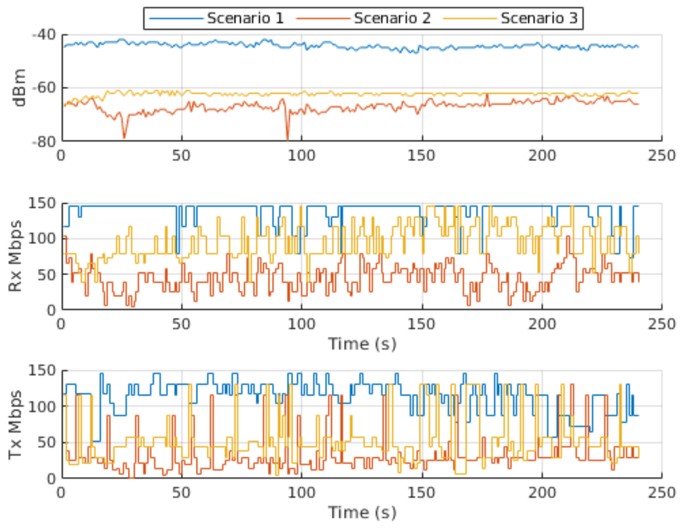


Figure 6.10: Signal strength & MCS bitrate experiment 4 - WiFi Network

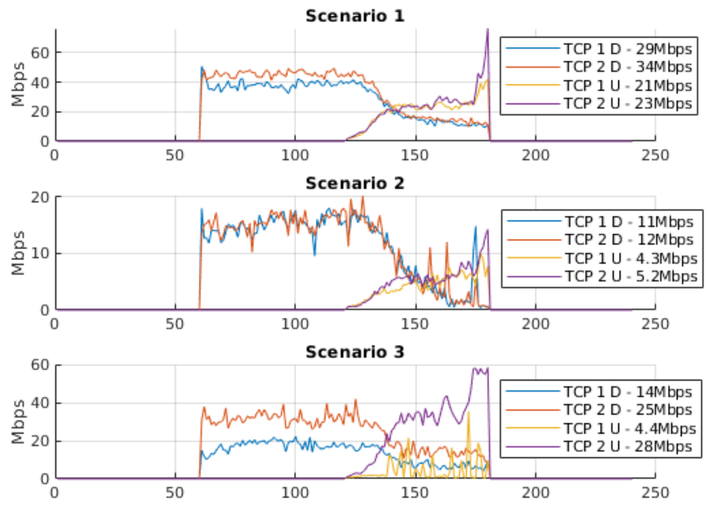


Figure 6.11: TCP rate Experiment 4 - WiFi Network (U = Uplink and D = Downlink)

Scenarios	Sec. 1	Sec. 2	Sec. 3	Sec. 4	Sec. 1	Sec. 2	Sec. 3	Sec. 4
Scenario 1	Uplink				Downlink			
GCC rate - mean (Mbps)	2.29	2.43	2.50	2.34	2.32	0.24	0.04	0.41
GCC rate - std (Mbps)	0.32	0.15	0.1	0.2	0.34	0.49	0.01	0.53
Delay - mean (ms)	3.67	231.63	445.44	73.46	3.67	231.63	445.44	73.46
Delay - 25th perc. (ms)	2.35	243.4	288.5	2.4	2.35	243.4	288.5	2.4
Delay - 50th perc. (ms)	2.8	273.7	497.1	5.3	2.8	273.7	497.1	5.3
Delay - 90th perc. (ms)	6.96	290.9	636.8	359.7	6.96	290.9	636.8	359.7
Delay - 95th perc. (ms)	8.39	296.8	674.3	552.2	8.39	296.8	674.3	552.2
Packet Loss - mean (%)	0.06	0.02	0.03	0.06	0	0.1	0.4	0.09
Packet Loss - max (%)	0.27	0.48	1.59	0.23	0	4.17	4.35	2
TCP 1 Rate (Mbps)	0	0	20.75	0	0	38.33	20.14	0
TCP 2 Rate (Mbps)	0	0	22.83	0	0	45.53	22.71	0
Scenario 2	Uplink				Downlink			
GCC rate - mean (Mbps)	1.76	1.34	1.34	1.89	2.34	0.21	0.04	0.47
GCC rate - std (Mbps)	0.27	0.21	0.25	0.26	0.33	0.47	0.01	0.51
Delay - mean (ms)	8.22	381.9	622.23	55.34	8.22	381.9	622.23	55.34
Delay - 25th perc. (ms)	4.7	302.9	485.9	7.8	4.7	302.9	485.9	7.8
Delay - 50th perc. (ms)	7	462	573.7	13.4	7	462	573.7	13.4
Delay - 90th perc. (ms)	14.12	528.1	971.5	199.4	14.12	528.1	971.5	199.4
Delay - 95th perc. (ms)	15.72	533.5	1039	251.5	15.72	533.5	1039	251.5
Packet Loss - mean (%)	0.02	0.01	0.15	0.24	0.05	0.37	1.33	0.28
Packet Loss - max (%)	0.12	0.62	5.1	6.68	1.34	4.55	9.76	4.26
TCP 1 Rate (Mbps)	0	0	4.28	0	0	15	7.6	0
TCP 2 Rate (Mbps)	0	0	5.15	0	0	15.36	7.65	0
Scenario 3	Uplink				Downlink			
GCC rate - mean (Mbps)	1.83	0.47	0.19	0.42	2.33	0.27	0.04	0.59
GCC rate - std (Mbps)	0.25	0.29	0.07	0.31	0.38	0.53	0.01	0.64
Delay - mean (ms)	16.31	265.98	479.23	184.89	16.31	265.98	479.23	184.89
Delay - 25th perc. (ms)	7.6	190.2	353.2	2.9	7.6	190.2	353.2	2.9
Delay - 50th perc. (ms)	11.8	291.2	417.7	10.5	11.8	291.2	417.7	10.5
Delay - 90th perc. (ms)	26.76	403.3	663.7	622	26.76	403.3	663.7	622
Delay - 95th perc. (ms)	44.91	427.3	915.8	845	44.91	427.3	915.8	845
Packet Loss - mean (%)	0.36	0.29	2.2	1.06	0.23	0.06	0.04	0.14
Packet Loss - max (%)	6.3	8.57	31.51	23.47	2.37	2.04	2.38	1.56
TCP 1 Rate (Mbps)	0	0	4.44	0	0	17.51	10.12	0
TCP 2 Rate (Mbps)	0	0	28.27	0	0	31.56	18.93	0

Table 6.7: Results TCP cross traffic experiment 4 - WiFi

Scenarios	Section 1	Section 2	Section 3	Section 1	Section 2	Section 3
Scenario 1	Uplink			Downlink		
GCC rate - mean (Mbps)	2.32	2.45	2.45	2.34	2.47	2.47
GCC rate - std (Mbps)	0.33	0.06	0.06	0.34	0.07	0.07
Delay - mean (ms)	22.66	22.01	22.65	22.66	22.01	22.65
Delay - 25th perc. (ms)	21	21.3	21.8	21	21.3	21.8
Delay - 50th perc. (ms)	22.4	22.2	22.5	22.4	22.2	22.5
Delay - 90th perc. (ms)	25.12	23.6	24.7	25.12	23.6	24.7
Delay - 95th perc. (ms)	25.66	23.6	25.6	25.66	23.6	25.6
Packet Loss - mean (%)	0	0	0	0	0	0
Packet Loss - max (%)	0	0	0	0	0	0
Scenario 2	Uplink			Downlink		
GCC rate - mean (Mbps)	2.3	2.45	2.45	2.33	2.46	2.41
GCC rate - std (Mbps)	0.35	0.05	0.05	0.33	0.07	0.09
Delay - mean (ms)	23.85	24.75	25.9	23.85	24.75	25.9
Delay - 25th perc. (ms)	22.8	23.6	24.9	22.8	23.6	24.9
Delay - 50th perc. (ms)	24	24.4	26	24	24.4	26
Delay - 90th perc. (ms)	26.6	27	28	26.6	27	28
Delay - 95th perc. (ms)	27.93	27.5	28.6	27.93	27.5	28.6
Packet Loss - mean (%)	0	0	0	0	0	0
Packet Loss - max (%)	0	0	0	0	0	0

Table 6.8: Results bidirectional experiment 1 - 4G Network

transmitting the uplink GCC call continuously found the channel busy. Therefore, the delay increased and GCC reacted by decreasing the sending bitrate.

With respect to the requirements of congestion control algorithms, GCC is not able to fulfill requirements 1c, 2 and 3 when the access technology is a 802.11n network. This is not caused by GCC itself but due to the random access technique (DCF) of this type of technology.

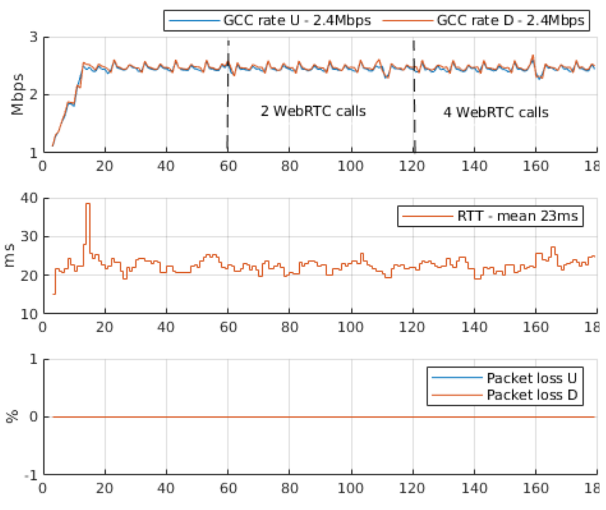
6.4. 4G Network

The results of the experiments obtained in a 4G network are presented in the following subsections. The main objective is to evaluate the performance of a WebRTC call when other GCC flows and TCP flows are competing for radio resources. This analysis focuses on the performance of an individual WebRTC call (GCC flow and video quality). GCC flows used to congest the radio links are not analyzed since their performance was limited by the CPU power of the laptops that handled these connections. Since the 4G testbed works in FDD (Frequency Division Duplexing), only bidirectional WebRTC calls are used in the experiments. All experiments evaluate the 2 scenarios described in chapter 5.4.

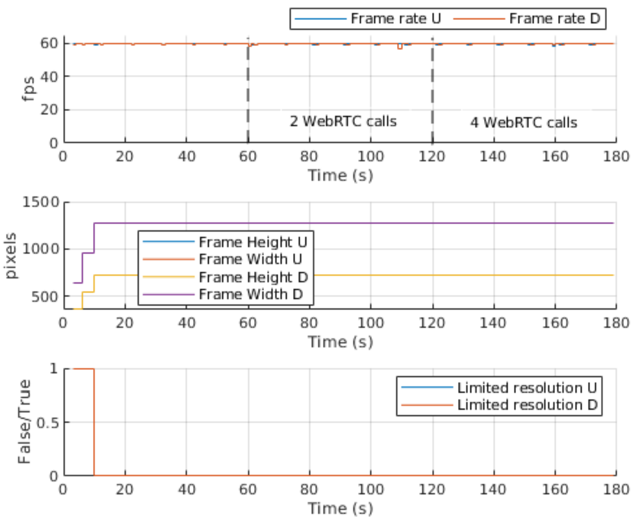
6.4.1. Competing Media Flows in a Bidirectional Transmission - Experiment 1

The aim of this experiment is to evaluate the performance of a bidirectional WebRTC call (uplink and downlink video calls) when other WebRTC calls are also present in the radio link. The experiment was divided in 3 sections of 60 seconds. In the first section, there was no competing traffic. In section 2, two bidirectional WebRTC calls were added. Finally, section 3 had four competing WebRTC calls. The evaluation of GCC is based on requirements 1a, 1b, 2 (partial) and 5. The results are presented in figure 6.12 and table 6.8.

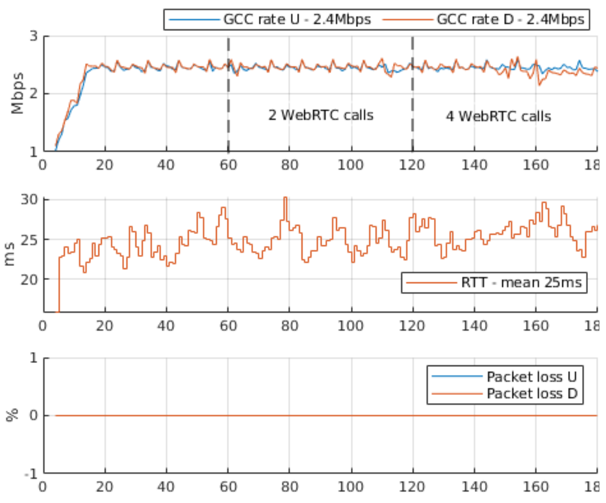
In scenario 1, neither the uplink nor the downlink GCC bitrates were affected by the addition of competing WebRTC calls. This can be observed in figure 6.12a. The bitrate in both directions was high and stable during sections 2 and 3. The mean bitrate was around 2.4 Mbps and the standard deviation was around 0.06 Mbps. The good quality of the radio links



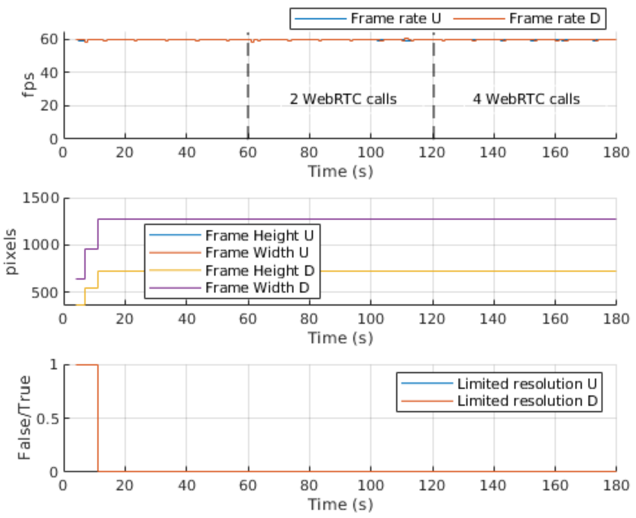
(a) Flow - Scenario 1



(b) Video - Scenario 1



(c) Flow - Scenario 2



(d) Video - Scenario 2

Figure 6.12: Graphs bidirectional experiment 1 - 4G Network (U = Uplink and D = Downlink)

Scenarios	Sec. 1	Sec. 2	Sec. 3	Sec. 4	Sec. 1	Sec. 2	Sec. 3	Sec. 4
Scenario 1	Uplink				Downlink			
GCC rate - mean (Mbps)	2.33	2.45	0.21	0.74	2.33	2.48	0.77	1.14
GCC rate - std (Mbps)	0.31	0.05	0.39	0.56	0.36	0.06	0.77	0.6
Delay - mean (ms)	22.29	24.68	1944.9	355.76	22.29	24.68	1944.9	355.76
Delay - 25th perc. (ms)	21.4	23.4	685.4	19.9	21.4	23.4	685.4	19.9
Delay - 50th perc. (ms)	22.6	24.4	2467.2	21.2	22.6	24.4	2467.2	21.2
Delay - 90th perc. (ms)	24.76	27	3167.3	1677.4	24.76	27	3167.3	1677.4
Delay - 95th perc. (ms)	25.93	27.7	3229.1	3286.8	25.93	27.7	3229.1	3286.8
Packet Loss - mean (%)	0	0	0	0	0	0	0	0
Packet Loss - max (%)	0	0	0	0	0	0	0	0
TCP 1 Rate (Mbps)	0	0	5.35	0	0	0	12.43	0
TCP 2 Rate (Mbps)	0	6.75	4.58	0	0	24.58	13.42	0
Scenario 2	Uplink				Downlink			
GCC rate - mean (Mbps)	2.32	2.44	0.21	0.95	2.31	2.47	0.48	0.62
GCC rate - std (Mbps)	0.33	0.05	0.43	0.67	0.34	0.07	0.72	0.52
Delay - mean (ms)	23.15	25.86	2028.4	374.95	23.15	25.86	2028.4	374.95
Delay - 25th perc. (ms)	22	24.63	713.88	21.25	22	24.63	713.88	21.25
Delay - 50th perc. (ms)	23.25	25.75	2587.8	22.75	23.25	25.75	2587.8	22.75
Delay - 90th perc. (ms)	26.25	27.75	3390.4	2007.2	26.25	27.75	3390.4	2007.2
Delay - 95th perc. (ms)	26.91	29.75	3550.2	3304	26.91	29.75	3550.2	3304
Packet Loss - mean (%)	0	0	0	0	0	0	0.02	0
Packet Loss - max (%)	0	0	0	0	0	0	1.09	0
TCP 1 Rate (Mbps)	0	0	4.47	0	0	0	6.23	0
TCP 2 Rate (Mbps)	0	6.81	4.65	0	0	10.88	7.83	0

Table 6.9: Results TCP cross traffic experiment 2 - 4G Network

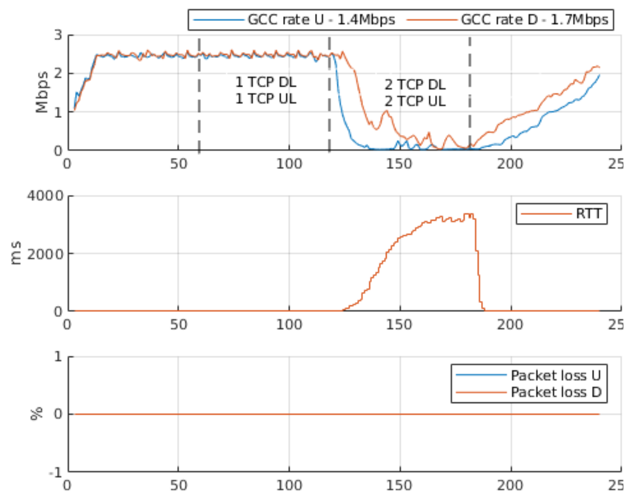
was also reflected with a low and stable delay (about 23 ms) and zero packet loss. With respect to the video quality, figure 6.12b shows that the frame rate was stable around 60 fps and the resolution was HD during the whole experiment.

The laptops in scenario 2 were located around 100 meters away from the antenna and around 90 degrees to the right of the main lobe of signal radiation. There were many obstacles between the laptops and the antenna such as trees, cars and a building. Visually, the phones showed only a single bar of signal strength. Despite these conditions, the communication was not degraded. Figures 6.12c and 6.12d show the good quality of the GCC flow and video stream. The results are similar to the ones obtained in scenario 1. This proves the great capacity and coverage of a 4G commercial antenna and therefore, the necessity to perform a good cell planning.

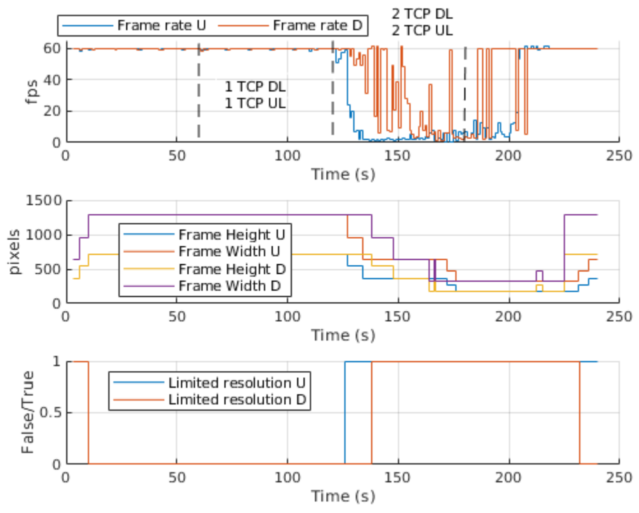
6.4.2. Media Flow Competing with a Long-lived TCP Flow - Experiment 2

In this experiment, long-lived TCP flows were used to congest the radio channel while a WebRTC communication was taking place. The experiment was divided in 4 sections of 60 seconds. In the first section, there was no competing traffic. Then, two TCP flows (one uplink and one downlink) were established between laptops PC2 and PC4 in section 2. In section 3, two additional TCP flows (one uplink and one downlink) were also added, this time between PC1 and PC3. Finally, in the last section, the TCP flows were removed. The evaluation of GCC is based on requirements 1, 2, 3 and 5. The results are presented in figure 6.13 and table 6.9. Figure 6.14 presents the TCP rates.

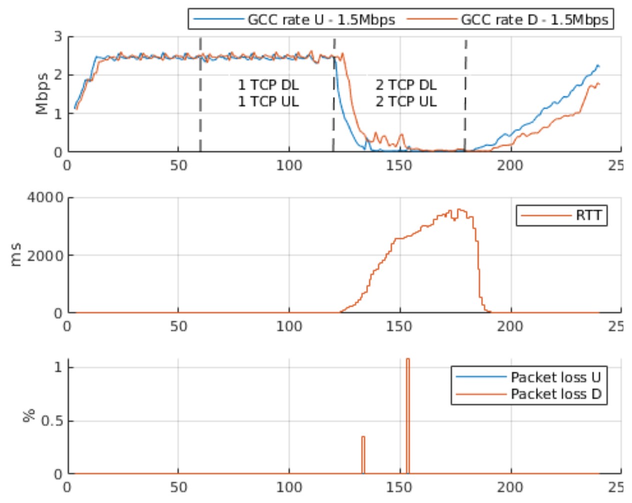
The GCC flow results of scenario 1 are presented in figure 6.13a. It can be observed that the bitrates were not affected by the TCP flows that were added in section 2. During this section, the bitrates were high and stable. The mean values were around 2.45 Mbps and the



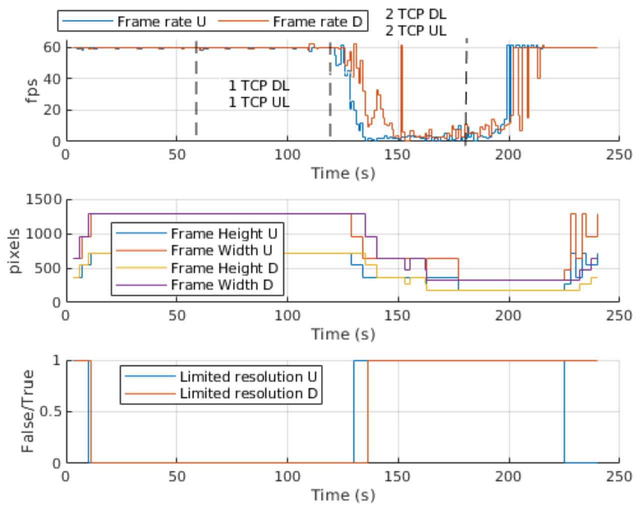
(a) Flow - Scenario 1



(b) Video - Scenario 1



(c) Flow - Scenario 2



(d) Video - Scenario 2

Figure 6.13: Graphs TCP cross traffic experiment 2 - 4G Network (U = Uplink and D = Downlink)

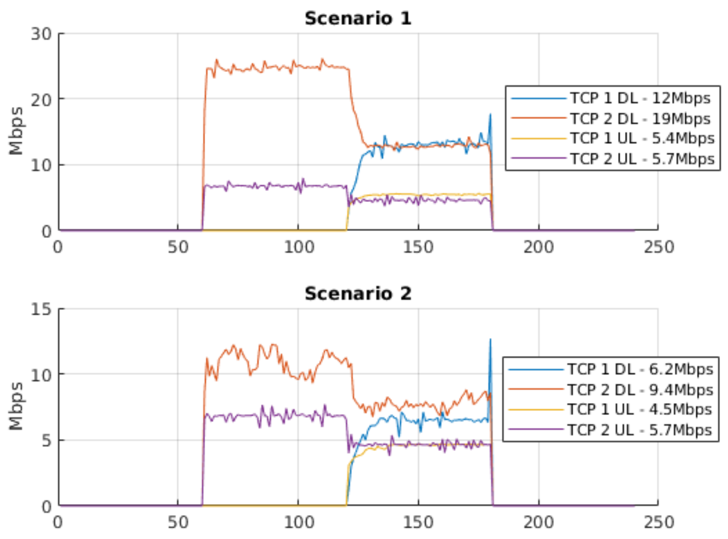


Figure 6.14: TCP rate Experiment 2 - 4G Network (UL = Uplink and DL = Downlink)

standard deviation was about 0.06 Mbps. The good quality of the connection is also reflected in the TCP rates. For the downlink TCP connection, the mean rate was 24.58 Mbps while the uplink TCP connection presented a mean value equal to 6.75 Mbps. When two additional TCP flows were added in section 3, the bitrates rapidly decreased. The delay was very high during this section. The mean value was about 2 seconds and the 95th percentile was 3.23 seconds. As expected, the video quality during this section was very poor as observed in figure 6.13b. The frame rate for both, uplink and downlink transmissions was very unstable and the resolution dropped to minimum values. WebRTC reported limited resolution due to bandwidth. The competing TCP flows were removed at 180 seconds, but GCC was only able to restore the connections around 225 seconds. This shows again the slow convergence time of GCC.

The results in scenario 2 are similar to the ones obtained in scenario 1. The delay was higher due to the channel conditions. In section 3, the 95th percentile of delay was 3.55 seconds. The TCP rates obtained in this scenario show that despite the location of the laptops, it was possible to reach a mean bitrate of 10.88 Mbps and 6.81 Mbps on the downlink and uplink connections respectively. This explains why it was not possible to congest the network with GCC flows in the previous experiment.

In a 4G radio network, the eNodeB performs channel-adaptive packet scheduling. The specific mechanism for scheduling is vendor specific. In general, there is a trade-off between efficiency and fairness. For instance, a scheduling technique that allocates radio resources to users with high Channel Quality Indicators (CQI) is very efficient since the channel utilization is high. However, users with low CQI (located at the edge of the cell) might never get channel resources. The details of the scheduling technique implemented by Ericsson was not revealed, however, it was indicated that the algorithm tends to be more fair than efficient. The packet scheduling decision is based on the CQI feedback, buffer status and QoS targets.

In this experiment, the first pair of TCP flows did not affect the GCC bitrates since these TCP flows were established by a different user. Even though, the TCP bitrates were much higher than the GCC bitrates, the packet scheduler made sure to fairly allocate resources to both users. A different situation occurred when the additional pair of TCP flows was established in the laptop that was also handling the GCC communication. The amount of TCP packets to be transmitted was higher than the amount of GCC packets. The TCP flows started to fill in the buffers in both directions causing an increase in delay. GCC reacted by decreasing its bitrates and the transmission finally collapsed. A possible solution to this issue would be to assign a different QoS class to the WebRTC video stream. This will implemented in a future work within Ericsson.

7 Conclusions and Future Work

WebRTC is seen by mobile operators and vendors as the key technology to enhance real-time communication services. However, it is not yet clear how WebRTC will interact with the IMS network nor what performance this type of service will have in real networks, especially when using wireless access technologies such as WiFi or 4G. This project addressed these issues by presenting a study on how WebRTC is being adopted by the mobile industry and performing an evaluation of the congestion control algorithm GCC in Ethernet, WiFi (802.11n) and 4G networks. The conclusions of this project, as well as, future work are presented below.

7.1. Conclusions

- Even though, the IETF has not yet selected a congestion control algorithm for WebRTC, GCC from Google is taking the lead in the market. It is the only congestion control algorithm for WebRTC that has been integrated in popular browsers such as Google Chrome and Firefox. Regarding the integration of WebRTC in mobile networks, specifically in the IMS network, 3GPP has imposed limitations to this technology which to our understanding is taking away the native capabilities and advantages of WebRTC. For instance, the standards do not allow the multiplexing of media streams in a single flow. An important aspect that has not been addressed by the 3GPP yet, is the definition of an architecture and procedures to provide QoS to the WebRTC video stream. This is crucial in applications that require high quality in mobile environments. From the industry side, companies like Ericsson are taking different approaches with respect to WebRTC. Some of these solutions fully integrate with the IMS network and follow partially the standards of the 3GPP. One main difference is that these solutions do not impose limitations on the native capabilities of WebRTC. Other types of solutions simply use IMS as a mechanism to trigger a WebRTC facilitator which is then in charge of establishing a WebRTC session over the normal Internet connection (default bearer). They see WebRTC as an over the top (OTT) service. It is evident that the 3GPP and vendors like Ericsson have different approaches regarding the integration of WebRTC in the IMS. Standardization bodies and industry still need to work on this topic to define a general architecture and procedures to accomplish this adoption.

- The evaluation of GCC in a wired network revealed that this algorithm has slow convergence when an increase in bitrate is needed. Even though, this has already been acknowledged by other studies, GCC has not yet been modified to solve this issue. On the other hand, the bitrate of GCC is very stable once it has converged. The experiments showed a small amplitude in the oscillations of bitrate caused by transitions in the GCC state machine. In general, there exists a trade-off between convergence and stability, the more aggressive the bitrate is increased, the less stable the bit rate becomes. However, since the increase state of GCC has two modes of operation, namely multiplicative and additive, it should be possible to find a mechanism to increase faster the bitrate in multiplicative mode and not modify the algorithm in additive mode to keep the same stability. Moreover, GCC has evolved since the time it was not able to share the link with both, real-time flows and loss-based flows. In a wired network, GCC is capable of fairly sharing the bandwidth with other media types while keeping the delay as low as possible.

- This project revealed that GCC is able to share the available bandwidth with a long-lived TCP flow (large file download) in an Ethernet connection (full-duplex with its own collision domain) but when it has to compete with this type of flow in a half-duplex wireless medium with high contention such as an 802.11n network, the bitrate collapses. This happens even when the channel conditions are good (low signal attenuation). This issue is mainly caused by the DCF protocol, which is the most common random access technique used in 802.11 networks. DCF tries to give equal access to the channel to all nodes in the network. Therefore, when the AP has more frames to transmit, there is an increase in delay since the channel allocation remains the same. The TCP flows do not adjust their bitrates due to packet loss because of the long buffers present in access devices. GCC simply reacts to this increase in delay by decreasing its bitrate expecting to reduce the delay. However, this never happens and the downlink bitrate is eventually reduced to zero. This behavior leads to the necessity of implementing mechanisms in the network to prevent the starvation of a WebRTC communication due to the presence of long-lived TCP flows. These mechanisms could interact with the GCC algorithm to be activated only when starvation is detected. In general, we believe that these mechanisms should assign a higher priority to the GCC traffic in order to speed up their transmission.
- The results obtained in a 4G network revealed that GCC successfully shares the available resources with long-lived TCP flows if they are being transmitted by other users. This behavior is achieved thanks to the packet scheduling mechanism of 4G networks. The scheduler makes sure to fairly divide the available resources among users. However, when long-lived TCP flows are being transmitted from the same terminal, the buffers start to fill up due to the high amount of packets generated by these TCP flows. The delay rapidly increases and this causes the collapse of the GCC communication. To avoid the starvation of GCC, it is necessary to explore procedures to assign a better QoS to the WebRTC communication. As it was mentioned previously, these procedures are yet to be explored in commercial solutions and defined by the 3GPP.

7.2. Future Work

Based on the conclusions of this work, the following activities are identified.

- Implement mechanisms to reduce the long convergence time of GCC. Study the implications that these mechanisms might have on bitrate stability.
- Study and implement mechanisms to avoid the starvation of GCC (or real-time flows in general) due to long-lived TCP flows in 802.11 networks. The implementation should focus on software and hardware modifications in the Access Point only. Perform an evaluation of GCC when competing with short-lived TCP flows (web page download) in 802.11 networks.
- Define architecture and procedures to provide QoS to the WebRTC communication following the 3GPP standards. Implement an interface to interconnect the WebRTC facilitator with the Policy and Charging Control (PCC). This interface must comply with the specifications of the Rx interface defined by 3GPP.

List of Figures

List of Tables

Bibliography

[1] Google. Webrtc, 2019. URL https://webrtc.org/.

[2] Sandvine. The global internet phenomenal report october 2018, October 2018. URL https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf.

[3] Ericsson. Ericsson mobility report november 2018, November 2018. URL https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-november-2018.pdf.

[4] IETF. *Congestion Control Requirements for Interactive Real-Time Media draft-ietf-rmcat-cc-requirements-09*, 06 2015. URL https://tools.ietf.org/html/draft-ietf-rmcat-cc-requirements-09.

[5] Luca De Cicco, Gaetano Carlucci, and Saverio Mascolo. Experimental investigation of the google congestion control for real-time flows. In *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, FhMN '13, pages 21–26, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2183-9. doi: 10.1145/2491172.2491182. URL http://doi.acm.org/10.1145/2491172.2491182.

[6] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, pages 13:1–13:12, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4297-1. doi: 10.1145/2910017.2910605. URL http://doi.acm.org/10.1145/2910017.2910605.

[7] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo. Congestion control for web real-time communication. *IEEE/ACM Transactions on Networking*, 25(5):2629–2642, Oct 2017. ISSN 1063-6692. doi: 10.1109/TNET.2017.2703615.

[8] IETF. *Test Cases for Evaluating RMCAT Proposals draft-ietf-rmcat-eval-test-10*, 05 2019. URL https://tools.ietf.org/html/draft-ietf-rmcat-eval-test-10.

[9] E. Janczukowicz, A. Braud, S. Tuffin, A. Bouabdallah, and J. Bonnin. Evaluation of network solutions for improving webrtc quality. In *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–5, Sep. 2016. doi: 10.1109/SOFTCOM.2016.7772141.

[10] B.A. Jansen. Performance analysis of webrtc-based video conferencing. Master's thesis, Delft University of Technology, 2016. URL https://repository.tudelft.nl/islandora/object/uuid:505300b0-f421-4470-b63e-cdb9154f7d54?collection=education.

[11] L. D. Cicco, G. Carlucci, and S. Mascolo. Congestion control for webrtc: Standardization status and open issues. *IEEE Communications Standards Magazine*, 1(2):22–27, 2017. ISSN 2471-2825. doi: 10.1109/MCOMSTD.2017.1700014.

[12] G. Carlucci, L. De Cicco, C. Ilharco, and S. Mascolo. Congestion control for real-time communications: A comparison between nada and gcc. In *2016 24th Mediterranean Conference on Control and Automation (MED)*, pages 575–580, June 2016. doi: 10.1109/MED.2016.7535978.

[13] L. Ma, W. Chen, D. Veer, G. Sternberg, W. Liu, and Y. Reznik. Early packet loss feedback for webrtc-based mobile video telephony over wi-fi. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2015. doi: 10.1109/GLOCOM.2015.7417847.

[14] Ingemar Johansson. Self-clocked rate adaptation for conversational video in lte. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14, pages 51–56, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2991-0. doi: 10.1145/2630088.2631976. URL `http://doi.acm.org/10.1145/2630088.2631976`.

[15] IETF. *Self-Clocked Rate Adaptation for Multimedia*, 12 2017. URL `https://tools.ietf.org/html/rfc8298`.

[16] IETF. *Overview: Real Time Protocols for Browser-based Applications draft-ietf-rtcweb-overview-19*, 05 2018. URL `https://tools.ietf.org/html/draft-ietf-rtcweb-overview-19`.

[17] W3C. *WebRTC 1.0*, 09 2018. URL `https://www.w3.org/TR/webrtc/`.

[18] IETF. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal*, 07 2018. URL `https://tools.ietf.org/html/rfc8445`.

[19] IETF. *SDP: Session Description Protocol*, 07 2006. URL `https://tools.ietf.org/html/rfc4566`.

[20] IETF. *Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol draft-ietf-ice-trickle-21*, 04 2018. URL `https://tools.ietf.org/html/draft-ietf-ice-trickle-21`.

[21] IETF. *RTP: A Transport Protocol for Real-Time Applications*, 07 2003. URL `https://tools.ietf.org/html/rfc3550`.

[22] IETF. *Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)*, 02 2008. URL `https://tools.ietf.org/html/rfc5124`.

[23] IETF. *The Secure Real-time Transport Protocol (SRTP)*, 03 2014. URL `https://tools.ietf.org/html/rfc3711`.

[24] IETF. *Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)*, 05 2010. URL `https://tools.ietf.org/html/rfc5763`.

[25] IETF. *A Google Congestion Control Algorithm for Real-Time Communication draft-ietf-rmcat-gcc-02*, 01 2017. URL `https://tools.ietf.org/html/draft-ietf-rmcat-gcc-02`.

[26] IETF. *NADA: A Unified Congestion Control Scheme for Real-Time Media draft-ietf-rmcat-nada-12*, 08 2019. URL `https://tools.ietf.org/html/draft-ietf-rmcat-nada-12`.

[27] IETF. *Stream Control Transmission Protocol*, 09 2007. URL `https://tools.ietf.org/html/rfc4960`.

[28] IETF. *Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets*, 11 2017. URL `https://tools.ietf.org/html/rfc8261`.

[29] IETF. *Stream Control Transmission Protocol (SCTP) Partial Reliability Extension*, 05 2004. URL `https://tools.ietf.org/html/rfc3758`.

[30] IETF. *WebRTC Data Channel Establishment Protocol draft-ietf-rtcweb-data-protocol-09.txt*, 01 2015. URL `https://tools.ietf.org/html/draft-ietf-rtcweb-data-protocol-09`.

[31] Ilya Grigorik. *High Performance Browser Networking*. O'Reilly Media, Inc, 2013. URL `https://hpbn.co/`.

[32] IETF. *Transports for WebRTC draft-ietf-rtcweb-transports-17*, 10 2016. URL `https://tools.ietf.org/html/draft-ietf-rtcweb-transports-17`.

[33] IETF. *WebRTC Audio Codec and Processing Requirements*, 05 2016. URL `https://tools.ietf.org/html/rfc7874`.

[34] IETF. *WebRTC Video Processing and Codec Requirements*, 03 2016. URL `https://tools.ietf.org/html/rfc7742`.

[35] IETF. *JavaScript Session Establishment Protocol draft-ietf-rtcweb-jsep-26*, 10 2017. URL `https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-26`.

[36] W3C. *Media Capture and Streams*, 10 2017. URL `https://www.w3.org/TR/mediacapture-streams/`.

[37] IETF. *Web Real-Time Communication (WebRTC): Media Transport and Use of RTP draft-ietf-rtcweb-rtp-usage-26*, 03 2016. URL `https://tools.ietf.org/html/draft-ietf-rtcweb-rtp-usage-26#page-11`.

[38] 3GPP. Universal Mobile Telecommunications System (UMTS); LTE; IP Multimedia Subsystem (IMS). Technical Specification (TS) 23.228, 3rd Generation Partnership Project (3GPP), 04 2019. Version 15.4.0.

[39] 3GPP. Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; Network architecture . Technical Specification (TS) 23.002, 3rd Generation Partnership Project (3GPP), 07 2018. Version 15.0.0.

[40] Catherine Mulligan Ioannis Fikouras Anders Ryde Mats Stille Rogier Noldus, Ulf Olsson. *IMS Application Developer's Handbook: Creating and Deploying Innovative IMS Applications*. Academic Press; 1 edition, 2011.

[41] 3GPP. Universal Mobile Telecommunications System (UMTS); Numbering, addressing and identification. Technical Specification (TS) 23.003, 3rd Generation Partnership Project (3GPP), 03 2019. Version 15.6.0.

[42] 3GPP. Universal Mobile Telecommunications System (UMTS); LTE; 3G security; Access security for IP-based services. Technical Specification (TS) 33.203, 3rd Generation Partnership Project (3GPP), 09 2018. Version 15.1.0.

[43] 3GPP. Universal Mobile Telecommunications System (UMTS); LTE; IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP). Technical Specification (TS) 24.229, 3rd Generation Partnership Project (3GPP), 04 2019. Version 15.6.0.

[44] 3GPP. Universal Mobile Telecommunications System (UMTS); Rx Interface and Rx/Gx signalling flows. Technical Specification (TS) 29.211, 3rd Generation Partnership Project (3GPP), 06 2007. Version 6.4.0.

[45] 3GPP. Universal Mobile Telecommunications System (UMTS); LTE; Web Real-Time Communications (WebRTC) access to the IP Multimedia (IM) Core Network (CN) subsystem (IMS). Technical Specification (TS) 24.371, 3rd Generation Partnership Project (3GPP), 06 2018. Version 15.0.0.

[46] IETF. *Evaluating Congestion Control for Interactive Real-time Media draft-ietf-rmcat-eval-criteria-09*, 02 2019. URL `https://tools.ietf.org/html/draft-ietf-rmcat-eval-criteria-09`.

[47] StatCounter. Browser market share worldwide, 2018. URL `http://gs.statcounter.com/browser-market-share/all/worldwide/2018`.

[48] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. pages 246–259, 2010. doi: Proc.ACMIMC.

[49] The Linux Foundation. netem, 2018. URL `https://wiki.linuxfoundation.org/networking/netem`.

[50] IETF. *Evaluation Test Cases for Interactive Real-Time Media over Cellular Networks Draft-sarker-rmcat-cellular-eval-test-cases-02*, 12 2014. URL `https://tools.ietf.org/html/draft-sarker-rmcat-cellular-eval-test-cases-02#section-1`.