A Framework and Evaluation of Rate Adaptive Video Telephony in 4G LTE

Nikolas Hermanns, Laurits Hamm, Ericsson GmbH, Eurolab R&D, Germany
Zaheduzzaman Sarker, Ericsson Research, Sweden
[nikolas.hermanns|laurits.hamm|zaheduzzaman.sarker]@ericsson.com

Abstract

4G LTE mobile networks are deployed by many operators and first operators offer telephony services over LTE (VoLTE) today. Video telephony is expected to be the next step. The required media bitrate for video is many times higher than for audio, which increases the risk of network congestion. Video bitrate adaptation techniques can avoid congestion. In this paper we evaluate the Google Congestion Control (GCC) algorithm and the Sprout algorithm, two recently proposed adaptation techniques. Based on the analysis of GCC and Sprout we propose an improved algorithm called E-Sprout. We present and discuss results from LTE system level simulations for video telephony using best effort bearers for the three algorithms.

Keywords – LTE, VoLTE, video, telephony, adaptation, Sprout, GCC

1 Introduction

Many operators around the world are rolling out 4G LTE (Long Term Evolution) mobile networks. At first LTE is mostly used to provide high speed mobile data. In a next step telephony services via LTE (VoLTE) can be offered using IMS and Multimedia Telephony services. First VoLTE deployments for voice are in operation today. VoLTE video is then the next step to offer video telephony via LTE and IMS [1].

VoLTE uses AMR-WB and H.264 codecs for audio and video respectively. The media bitrate for video is many times higher than for audio. If we assume a bitrate of 20 kbps for AMR-WB audio and a bitrate of 400 kbps for H.264 standard definition video, the required bitrate for video is 20 times higher than for audio. The majority of mobile devices supporting LTE also support high definition (HD) video. For HD video at a good quality one should assume a minimum bitrate of 1 Mbps and higher. Next generation video codecs like H.265 (HEVC) are expected to achieve the same video quality at almost half of the bitrate. Still, the required bitrates for video will be high and likely more efficient video codecs will not only be used to reduce bitrates, but also to increase video resolution and video quality.

Video sessions easily use up the available bandwidth in LTE. Multiple users use the available capacity in an LTE cell at the same time and share the same radio resources. In a realistic scenario different types of services over LTE are being used at the same time, e.g., telephony, web browsing and streaming. Those two factors lead to a changing available bitrate per user over time. Mobility of users is yet another source of available bitrate variation.

If the overall load in a radio cell increases and reaches the cell capacity, congestion occurs and timely transmission of data packets is not possible. It is the task of the radio scheduler to allocate radio resources to the users according to their service profile. If a timely transmission of packets is not possible then they can either be delayed or dropped. In packet based transmission the packets are typically queued, if they cannot be transmitted at the current time. Queues can then be emptied at a later point. Queuing adds delay to the transmission. If the maximum queue size is reached, packets will be dropped. Active queue management (AQM) is a technique to drop packets before the queue is full. We do not use AQM in this evaluation.

Real-time conversational services are very delay critical, meaning that the user experience will degrade largely if the end-to-end delay or one way delay (OWD) increases too much. Acceptable one way delays for voice telephony are around 200 ms (mouth to ear). Acceptable end-to-end delays for video telephony are higher around 400 ms [2]. Still, this delay puts a very strong requirement on the transmission for video.

A video stream is more likely to experience congestion due to the high video bitrate. To avoid too long one way delays in case of congestion, it is desirable to adapt and reduce the bitrate according to the available bitrate. If the available bitrate increases the video rate can be increased to send video at a higher quality. Thus, rate adaptation serves to send video with the best possible quality and to avoid too long delays.

The 3GPP Quality of Service (QoS) architecture for LTE and EPC (Evolved Packet Core) uses different bearers to handle different traffic types according to their QoS classes. Different bitrate and delay requirements are used for example for real-time audio and web browsing. Real-time conversational audio and video streams use dedicated

bearers, for which a guaranteed bitrate, a maximum bitrate, delay and packet loss thresholds are defined. Even within this framework and using dedicated bearers congestion for high bitrate video can occur. To keep an initially high video bitrate for a user will be very expensive and will consume a big part of the radio resources, if the link conditions for this user become worse. 3GPP defines the usage of explicit congestion notification (ECN). ECN requires managed network access. Therefore, we do not use ECN in this evaluation.
Recently the usage of best effort bearers, which do not guarantee a bitrate, delay or packet loss thresholds, for audio and video has drawn attention. Best effort bearers are used for RCS telephony, by Over-The-Top services, and in cases of unmanaged access networks, e.g., Wi-Fi. A very low fixed video bitrate and quality would have to be used to guarantee timely transmission using best effort bearers. Bitrate adaptive video on the other hand allows using best effort bearers and the maximum possible video bitrate and quality. We use best effort bearers in this work to evaluate the proposed adaptation algorithms for video in LTE.

Chapter 2 presents the adaptation framework and the adaptation algorithms: GCC, Sprout and E-Sprout. Chapter 3 describes the simulation setup and performance indicators used to evaluate the algorithms. Chapter 4 presents and discusses the simulation results.

2 Adaptation Framework and Algorithms

Figure 1 shows the media rate adaptation framework. It is implemented in the clients and splits up in a sender and a receiver part.

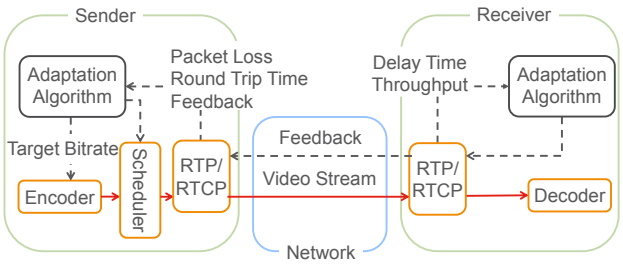

Figure 1 Adaptation framework showing sender and receiver components.

Different channel metrics are measured to detect congestion, e.g., throughput, round-trip time (RTT), packet loss rate and the time between two arriving video frames. The receiver estimates the channel and sends back a feedback to the sender. The sender is the controlling part in the framework and sets the target bitrate for the encoder.
In case of Sprout and E-Sprout, packets are queued in the scheduler block in order to protect the channel from congestion. The transmission of the video is done by the Real-Time Transport Protocol (*RTP*). For GCC the feedback is sent over Real-Time Transport Control Protocol (RTCP) reports. For Sprout and E-Sprout the feedback is piggy-backed on the media packets.

2.1 Adaptation Framework for Sprout and E-Sprout

Figure 2 shows how Sprout and E-Sprout fit in the adaptation framework. After detecting congestion in the channel the algorithms buffer the outgoing media flow at the sender. It also controls the media source by setting the target video bitrate considering the congestion in the channel.

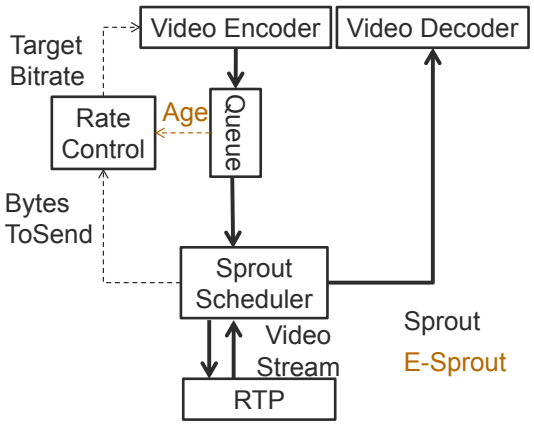

Figure 2 Sprout adaptation framework and E-Sprout in red.

The encoder produces a $1/F$ periodic data stream, where F is the frame period of the encoded video. If no congestion is detected, Sprout will fetch the video frames from the queue, packetize them and send them to the network socket which transfers the packets over the radio link. On the receiving side Sprout and E-Sprout measure the incoming bitrate, de-packetize the packets and delivers them to the video decoder. After the post calculations of the incoming data bitrate, described in Section 2.2, the information about the channel is sent back to the sender, which adapts its sending rate and the target bitrate from the encoder.
In addition, E-Sprout measures the RTT and uses the age of the oldest data in the queue to adapt the target bitrate.

2.2 Sprout

Sprout strives for the highest possible throughput, while preventing packets from waiting longer than 100 ms in the network queues [3].
Sprout measures the received throughput and forecasts it to a maximum of 160 ms window. The Sprout receiver uses a model of the channel which consists of a queue and a processing entity. The throughput of the entity is modeled by a Poisson process, whose rate varies with Browian motion. The Sprout receiver forecasts the throughput of the channel in periods of 20 ms (called “tick”). It forecasts the channel throughput each tick in such a way that packets

are delivered through the channel within 100 ms with a probability of 95 %. The sender receives this feedback from the receiver and calculates the *cumulative_delivery_forecast*.
The amount of bytes, which are save to be sent in the recent tick, are called *BytesToSend.* In order to calculate *BytesToSend,* Sprout estimates the number of packets which already arrived at the receiver and subtracts those from *cumulative_delivery_forecast*. To be able to do so the number of arrived bytes at the sender is added to the feedback.
With each received forecast the receiver's tick is set back to zero. If the sender gets no feedback for 160 ms, Sprout will reduce its sending rate to a heartbeat sending one packet every 50 ms.

2.3 E-Sprout

E-Sprout is an evolved version of Sprout which improves Sprout for video traffic. In E-Sprout the rate control estimates the target bitrate from both *BytesToSend* and the age of the oldest packets in the sending queue. If packets wait longer than the frame period, the target bitrate will be decreased.
For the rate control, *BytesToSend* is very unstable and has to be filtered. E-Sprout filters *BytesToSend* through a moving average and then again through an exponential filter. The moving average filter reduces peaks. The exponential filter provides smoothness when sudden changes of *BytesToSend* occur. The numbers of bytes, which are queued for transmission, are subtracted from *BytesToSend*. The filtered *BytesToSend* still varies in small scale. To prevent the encoder from changing the bitrate too frequently, the minimum time between two changes is set equal to the RTT.
At the beginning of a session, Sprout is quite optimistic and sends a high data rate, although it has not yet received a forecast and has no knowledge of the channel condition. To prevent congestion in this time E-Sprout sends the minimal data rate until the second forecast from the receiver has arrived.
As mentioned in Section 4.1 a problem of Sprout is the wrong detection of congestion. To handle this, E-Sprout adds the RTT of the channel to the estimation. If the RTT is less than 100 ms, the channel is not congested. Thus Sprout measures the video bitrate instead of the available channel bitrate. In this period of time E-Sprout uses one of the last eight forecasts, which promises the highest bitrate. This ensures that the algorithm achieves the highest possible bitrate, if the channel is not congested.
In Sprout every sent packet has a fallback time, marking the time when the next packet should arrive at the receiver. It is known that there will be no data to send between video frames, so the time, until the next frame will be produced by the encoder, is added to the fallback time.
The age of the packets, which are waiting for delivery, can be seen as pre-delay before transmission. This time is directly added to the one way delay and downgrades the user experience. To prevent a high pre-delay the packets are not queued up longer than two times the frame period. After that they are directly sent, although E-Sprout is estimating bad channel congestion.

2.4 Google Congestion Control

Google proposed the Google Congestion Control (GCC) in IETF (RMCAT WG) for real time conversational media. GCC is part of their WebRTC implementation code [4].
Like in Figure 1 the adaptation framework for GCC consists of a sender side controller and a receiver side measurement block. The receiver side controller takes the *inter-frame-arrival-time,* described in Section 2.4.1, to estimate the available channel bitrate. The sender side measurement block takes the packet loss rate and the receiver side estimated available bitrate to calculate the target bitrate of the encoder.
GCC uses the RTCP reports to send a rate request and packet loss rate from the receiver to the sender. The scheduler block shown in Figure 1 is not used by GCC. The frames generated from the encoder are directly processed by the RTP stack and sent to the network.

2.4.1 Receiver-Side Control

GCC measures the length of the arriving frames and the *inter-arrival-time*, which is the time between two arriving frames, to estimate the available bitrate of the channel. In order to do so the capacity of the channel is calculated through a Kalman filter.
GCC feeds the outcome m of the Kalman filter to its over-use detector. This detector has three states: over-use, under-use and normal. The states are triggered depending on the value of m.
The rate control estimates the available bitrate $\hat{A}$ of the channel. Depending on the state of the over-use detector the estimation $\hat{A}$ will be increased or decreased, shown in Figure 3. If the rate control is in hold state the estimated bitrate will be kept equal. If an under-use state is detected, the system will stay in the hold state until m is stabilizing again. Then the normal state will be triggered and the estimated available bitrate will be increased again. This helps GCC to clear the network queues after congestion.

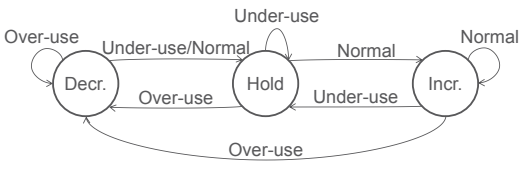

Figure 3 State chart of rate control with signals from over-use detector [4].

2.4.2 Sender-Side Control

The receiver sends back the estimated available bitrate to the sender. With this information, the packet loss rate and the RTT, the sender estimates the available bitrate $\hat{A}_s$. Three possible situations are taken into account. If the

packet loss rate p is between $2-10\%$ the estimation will be kept equal. If it is above 10% the new estimation will be $\hat{A}_s(i) = \hat{A}_s(i-1) * \left(1 - \frac{p}{2}\right)$. If p is under 2% it will be $\hat{A}_s(i) = 1.05 * \left(\hat{A}_s(i-1) + 1000\right)$. The estimation is bounded by the TCP friendly rate and the receiver estimated available bitrate.

3 Simulation and Evaluation

3.1 LTE System Level Simulation

The evaluation of the algorithms is done through a multi user radio system simulator with detailed models for the LTE physical layer, protocol layers, and radio resource management.

The algorithm framework is embedded into the simulated mobile user equipment. For each session a mobile user is connected with a fixed user on the network side, so that there is only one radio interface for uplink and downlink respectively. Only downlink simulations are performed so that the video is only send from fixed user to mobile user.

The video used for the simulations is a constant bitrate video with a nominal bitrate of 1.5 Mbps. The video source is rate adaptive between 0.15 to 1.5 Mbps and provides frames periodically with 30 fps. The length of each call is limited to 60 seconds.

3.1.1 Scenarios

Two scenarios are used in the simulations. The First scenario simulates an uncongested network scenario, which is called "bitpipe" in this paper.

The second scenario is an LTE radio system simulation, which is a 3GPP Case 1-like scenario with 21 cells, 5 MHz bandwidth, a 2 GHz carrier frequency, and SIMO antenna structure [5]. The users are distributed randomly over the simulation area and move in straight lines with random directions and a speed of 3 km/h. A simplified handover algorithm with ideal measurements is used. The variation of available bitrate for each user is done by the number of users system wide, so that with more users the cell will be more congested. The arrival of users follows a Poisson distribution, which is proportional to the length of the call, so that on average the number of users per cell is kept equal. Users, who transmit files over TCP, are added as background traffic. These users are called FTP users. Because the file size is rather short and FTP users arrive through a Poisson process the system is frequently congested. The average FTP load is 2 Mbps. A proportional fair scheduler allocates the bandwidth for each user. The video and FTP users are on the same bearer with the same priority, meaning that both session types are scheduled over best effort bearers.

3.2 Key Performance Indicators

Key performance indicators (KPI) describe specific characteristics of the algorithms or radio system. Channel metrics are measured and then evaluated to produce the KPIs. The main purpose of these indicators is to provide a comparison independent from design and technologies.

3.2.1 User Video Bitrate

The video bitrate x per user is the number of bits per second transmitted from the video sender to the video receiver..

3.2.2 One Way Delay

The one way delay (*OWD*), which is also known as the end-to-end delay, is measured from camera to screen which means to measure the time between the recording of the video and its presentation.

In [2] it is said, that the *OWD* should be kept well within the acceptable levels of 150 ms and must not exceed 400 ms. In [6] it is said that a OWD of 350 ms is still acceptable and is produced by commercial videoconferencing systems.

3.2.3 Radio Subband Utilization

In mobile networks one target is to completely utilize the available spectrum or radio resources. The maximum available link bitrate is mostly depending on the location of the users. If all users are in rather bad conditions, e.g. shadowing or long distance paths, the system has a low cell throughput, although the utilization of the network is high. To bring this into a KPI the subband utilization can be used, which is the percentage of used parts from the complete frequency band. For LTE the number of used subbands per user varies. So, a user can allocate more than one subband. If all subbands are used the system is completely utilized.

3.2.4 Algorithm Stability

Stability is associated with video bitrate variations. To measure per-session rate variations the coefficient of variation (*CoV)* is used. Let's assume for an interval $[1, m]$ that $\bar{x}$ is the average and σ the standard deviation of per-session video bitrate. Then the coefficient of variation is defined as:

$$CoV = \frac{\sigma}{\bar{x}} = \frac{\sqrt{\frac{1}{m}\sum_{i=1}^{m}(x(i) - \bar{x})^2}}{\frac{1}{m}\sum_{i=1}^{m} x(i)}. \quad (1)$$

The smaller the stability index is, the less oscillation a source experiences. The frame loss rate is related to the stability of the OWD and the throughput and will be minimized if the algorithm is more stable [7].

4 Results and Discussion

Figure 4 shows the simulation results for the KPIs described in Section 3.2. We plot the results for the three adaptation algorithms and a reference case for a minimum video bitrate of 150 kbps, where the video bitrate was constant throughout the video session.

GCC and Sprout are not performing well in the conversional video scenario. Both algorithms do not adapt to the available bitrate. GCC does not keep the delay under 400 ms. The amount of users which can be served with a GCC controlled media session is much lower than a session controlled by E-Sprout.

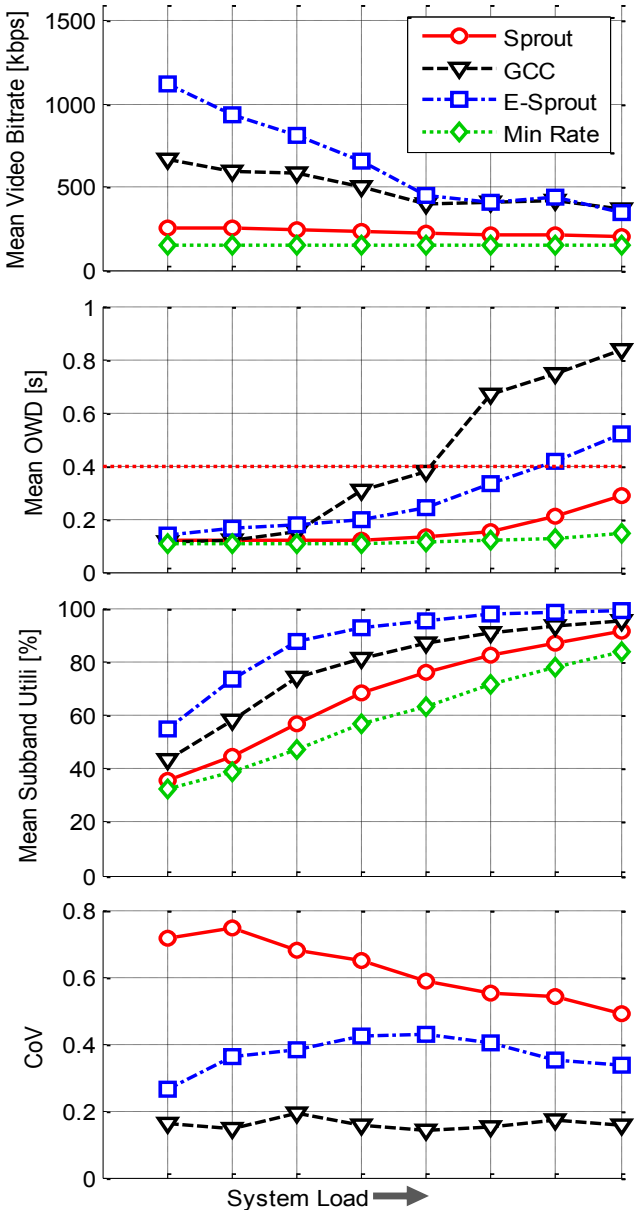

Figure 4 Simulation results for the different adaptation algorithms. A minimum fixed video bitrate of 150 kbps is used as a reference.

GCC is very conservative at low load of the system and achieves a lower video bitrate than E-Sprout. As the system load increases many of the users using GCC have an unacceptable high OWD, although the system is not fully utilized. This leads to the conclusion that GCC does not adapt fast enough. Due to the fact that it is much more constant, which can be seen by the CoV, it fails to cope with frequently changing channel conditions.

In our simulations with bulk transfer we see that Sprout works fine. But in the conversional video simulation the video bitrate of Sprout is almost equal to the minimum bitrate and does not fully utilize the network. A possible explanation is given in Section 4.1. The utilization is nearly proportional to the system load. The OWD plot shows that Sprout adds delay although it only sends a low bitrate. It queues up packets because of false congestion detection and adds unnecessary delay. Sprout has a very high CoV and is very unstable in the conversional video scenario.

The CoV for E-Sprout is medium high, which is achieved by applying the filtering of *BytesToSend*. As intended, the amount and the ratio of changes are reduced, compared to Sprout. It keeps the delay at the desired level while achieving high bitrates. Only at very high load the OWD increases. At this load the video rate is expected to converge to the minimum bitrate. At low load E-Sprout achieves the highest video bitrate and is more conservative in high load, which in comparison to GCC results in a much lower OWD.

The radio subband utilization shows that E-Sprout utilizes the network more than all other algorithms. The reason is the ability of E-Sprout to measure the channel directly. Compared to GCC, which probes the channel by constantly increasing the bitrate until the available bitrate of the channel is reached, E-Sprout can directly probe the channel without losing time, if there is enough data to send. This ability is taken form the original algorithm. E-Sprout sends such a high amount of data that the channel is directly utilized, while preventing congestion. By measuring the incoming throughput E-Sprout estimates the channel and directly adjusts the video bitrate. This makes the algorithm much faster at the beginning of the call and after congestion. In our simulations E-Sprout reaches the available bitrate up to 20 times faster than GCC. In the conversional video scenario a conclusion for Sprout cannot be given because it does not reach the available bitrate.

The CoV is useful to compare the algorithms, but it does not give an absolute answer to the algorithm stability. On one hand an algorithm should adapt fast enough to follow the available bitrate, on the other hand, the algorithm should not be unstable through false congestion detection. A preferred range of CoV cannot be given.

4.1 Sprout and Conversational Video

Figure 5 shows the inner algorithm metrics of Sprout at the beginning of a call in the scenario bitpipe, where the algorithm works in an uncongested channel. In the upper graph the data which can be sent is limited by the encoder. In the bottom graph the data is unlimited. In this setup the data, which is sent by the algorithm, is not limited and random data from the video encoder is sent.

One can see that the amount of data which Sprout wants to send at the beginning is high, although Sprout has not received any feedback yet. Since the available bitrate is not yet estimated, this may lead to high delay and packet loss, if the algorithm starts in a congestion scenario.

The amount of packets produced by the encoder is limited and periodic. This leads to the fact that *BytesToSend* and *BytesSent*, which is the actually amount of data transmitted to the radio link, are different in the upper graph of Figure 5. Therefore, the receiver measures the limited video bitrate instead of the available channel bitrate and detects false congestion. The result is an unstable behavior of Sprout which causes the very high CoV, see Figure 4.

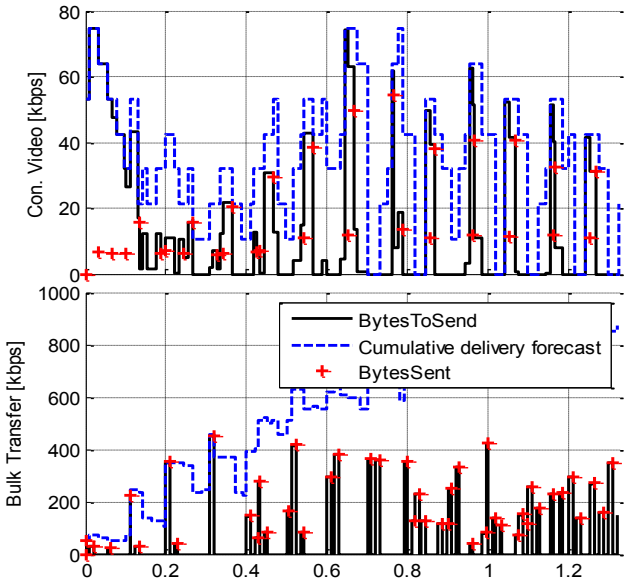

Figure 5 Sprout's inner algorithm metrics. *BytesSent* is the amount of data actually transmitted. *Cumulative delivery forecast* is the channel estimation. *BytesToSend* is the amount of data save to transmit in this tick.

In the unlimited case in the lower graph of Figure 5 *BytesToSend* and *BytesSent* are equal. Hear the amount of measured data is equal to the available bitrate of the channel.

5 Conclusion and Outlook

We evaluated the adaptation algorithms GCC and Sprout for real-time video over LTE. The results show that both algorithms do not perform well in this case. The improved E-Sprout algorithm shows a better and more stable performance, keeping the end-to-end delay at acceptable levels while achieving high bitrates at the same time. In a highly loaded system E-Sprout is still not working satisfactory enough. This might be a result of policies taken to handle low load situations. However, performance compared to GCC and Sprout is good.

The traffic characteristics of real-time video play a crucial role in designing an adaptation framework and algorithm for video telephony. It is the nature of video traffic that data is available periodically according to the video frame rate. Also the size of encoded video frames varies greatly. This challenges the congestion detection and the estimation of the available bitrate on the channel by throughput measurement only. RTT or OWD as an additional metric should be used.

E-Sprout and Sprout show the idea of fast adaptation through data burst measurements to fully utilize the network, while being careful not to send too high bursts. This gives a direct guess of the channel conditions and allows faster adaptation.

In future work more scenarios can be considered, e.g., uplink simulations. The convergence of the video bitrate to minimum bitrate using E-Sprout in high load situations needs further investigation. Another area of interest is the use of explicit congestion notification (ECN). The presented adaptation framework is implemented in the clients only, which have no knowledge about the other users in the cell. Using explicit congestion information from the network and base station can possibly lead to further improvements.

6 Acknowledgement

The authors thank their following colleagues for valuable feedback during this work: Ingemar Johansson, Thorsten Lohmar, Markus Andersson and Tomas Frankkila. We also thank Christian Feldmann from RWTH Aachen University (IENT) for fruitful discussions.

7 Literature

[1] GSMA IR.94, IMS Profile for Conversational Video Service.

[2] International Telecommunications Union, "One-way transmission time," in ITU-T Recommendation G.114, May 2003.

[3] Wistein, A. Sivaraman and H. Balakrishnan, "Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks," in 10th USENIX Symposium on Networked System Design and Implementation, NSDI'13, Lombard, 2013.

[4] H. Lundin, S. Holmer and H. Alvestrand, "A Google Congestion Control Algorithm for Real-Time Communication - draft-alvestrand-rmcat-congestion-00," 18 February 2013.

[5] 3GPP, "Physical layer aspects for evolved Universal Terrestrial Radio Access (UTRA)," Tech. Rep. 3GPP TR 25.814 V7.1.0, (Release 7) (2006-09), 3GPP, 2006.

[6] Y. Xu, C. Yu, J. Li and Y. Liu, "Video Telephony for End-consumers: Measurement Study of Google+, iChat, and Skype," in ACM conference on Internet measurement, Boston, Massachusetts, USA, 2012.

[7] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms RFC 5166," March 2008.