Interactive RTP services with Predictable Reliability

Jochen Gruen, Manuel Gorius, Thorsten Herfet
Saarland University
{gruen, gorius, herfet}@cs.uni-saarland.de

***Abstract*—Despite the fact that non-interactive Internet multimedia applications are prevalently deployed via HTTP/TCP, dedicated real-time protocols remain the essential basis for interactive voice and video services. This is particularly pronounced by the standardization of WebRTC, which enables peer-to-peer real-time communication between web browsers via the RTP protocol suite. However, a compound solution that contributes error and congestion control for RTP-based communications services is missing by now. In our previous work we presented a novel transport protocol – *Predictably Reliable Real-time Transport* (PRRT) – a protocol layer that efficiently supports the reliability required by interactive multimedia services under their specific time constraint. In the following paper we integrate PRRT's workflow and header format into a bi-directional RTP session by enhancement of the profile for generic forward error correction. We particularly focus on a low messaging overhead and the preservation of backward compatibility in the protocol's error control.**

***Index Terms*—RTP, WebRTC, Error Control, Congestion Control, Predictable Reliability**

I. Introduction

The convenience of integrating audio and video objects into HTML-based (Hyper Text Markup Language) web design renders the web browser the prevalent sink for non-interactive audiovisual multimedia streams. As a result of recent standardization activities in the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF), Web Real-Time Communication[1] (WebRTC) drafts an interface that provides peer-to-peer real-time browser communication in a similarly convenient fashion. The upcoming standard is expected to facilitate vast numbers of interactive communications services via the web browser.

A. Problem Statement

Unlike non-interactive web services, WebRTC builds upon the Real-time Transport Protocol (RTP) [9], which is desirably implemented on top of the User Datagram Protocol (UDP). As UDP only offers best-effort packet delivery, WebRTC applications are facing two fundamental problems imposed by today's Internet infrastructure. First, the heterogeneity of Internet paths, including both wired and wireless network segments, exposes upper-layer protocols to highly variable delay and packet loss characteristics that need to be addressed with suitable reliability schemes. Second, the self-managing rate allocation policy in the Internet requires transport protocols to implement congestion control in order to approach a fair share of throughput and to ensure overall network stability.

[1]http://www.w3.org/2011/04/webrtc/

B. Related Work

The RTP profile for generic forward error correction (RTP/FEC) [5] and the extended RTP profile for Real-time Transport Control Protocol (RTCP)-based feedback (RTP/AVPF) [8] have been proposed to improve the reliability of interactive communications services. RTP/FEC applies upper layer Forward Error Coding (FEC) to the RTP source packets without mentioning a specific adaptation algorithm. Without adaptation, packet-level FEC might result in significant coding overhead under variable network conditions. RTP/AVPF integrates negative acknowledgments (NACK) into RTCP receiver reports. However, this scheme cannot satisfy the reliability requirements of time-critical communications services as the feedback on packet losses is highly restricted under RTCP's scalability rules.

Besides error control, congestion control is similarly important for multimedia streams in order to ensure fair rate allocation in the Internet. Rate control of inelastic, audiovisual real-time streams is, however, a challenging topic. A recently published Internet draft [4] states clear requirements for the design of congestion control algorithms that are suitable for interactive RTP services. As a result of the IETF activities around RMCAT, several drafts propose media-friendly congestion control algorithms [10], [6], [7].

C. Contribution

Under the assumption that interactive real-time communication prefers timeliness over reliability, we have introduced the idea of *predictable reliability* within previous work [3]. This paradigm has been implemented into a new protocol design – the Predictably Reliable Real-time Transport protocol (PRRT) [2]. In order to predictably achieve the desired level of reliability, proactive and reactive error control are being optimized under the application's delay constraint. Predictably reliable error control relies on stochastic modeling of the protocol's reaction to the modeled packet loss behavior of the network path. The result of the joined modeling is periodically evaluated by a reliability control policy that validates the protocol configuration under the application constraints and under consideration of the available network bandwidth. The protocol applies an adaptive Hybrid Error Coding (HEC) approach that flexibly composes reactive and proactive error control. These schemes have been found to lead to near-optimum coding efficiency. PRRT's performance model optimizes error control under equation-based congestion control such that the protocol can conveniently be extended by various congestion control algorithms [2].

II. Protocol Specification

A. Message Format

PRRT shares several header fields with RTP. In particular it enhances UDP and requires sequence numbers as well as packet timestamps for packet loss detection and receiver synchronization, respectively. The time-constrained error control operates according to an optimized schedule (Figure 4) that requires a set of packet-specific timers to be communicated to the receiver. Further, the block coding parameters of the adaptive HEC scheme must be announced along with the transmission of coded repair packets.

In order to implement PRRT's feature set with maximum standard compliance and backward compatibility, we propose an RTP header extension and an additional repair packet channel based on the RTP profile for Generic Forward Error Correction [5]. Similarly to RTP/FEC, we establish two separate RTP sessions, whereas one transmits media data and a second session adds repair packets that are generated by a packet-level block-erasure code. Both sessions instantiate bidirectional, point-to-point connections as specified by the WebRTC draft for interactive real-time communication [1]. In contrast to RTP/FEC, our RTP profile does not preclude the usage of feedback. The parameter adaptation of the adaptive HEC requires frequent updates on the network path's Packet Loss Rate (PLR) and Round Trip Time (RTT) as well as a signaling of lost packets. Network state feedback and NACKs are therefore "piggybacked" to the media packets of the opposite RTP flow.

The first RTP session transmits media packets under backward compatibility and in compliance with RFC 3550 [9]. Thereby, RTP clients without support for the presented profile are enabled to receive the media packets passively without performing the error control. Media packets contain media data as defined in RTP's audio visual profile. The standard RTP header is followed by a header extension that contains indexing and timing information for the HEC scheme (Figure 1). In addition, as a feature of the bidirectional RTP session, network state feedback and packet loss notifications for the respectively opposite RTP flow are optionally included into the header extension. In case of low media packet frequency, urgent feedback is sent without a media payload.

In detail PRRT requires the RTP extension to carry the following header fields:

- The *index* field determines the position of the media packet in the packet-level block-erasure code.
- *RTT probe* updates the RTT estimate at the sending peer.
- The *length* of the header extension, including timer fields (*packet timeout*, *decoding timeout*, *feedback interval*), network state feedback and NACKs.
- Optional network state feedback is signaled by *F=1* and appended via the fields *PLR* and *bandwidth estimate*.
- In case the receiver observes packet losses, a variable number of *NACKs* identifying the erased packets terminates the header extension.

The second RTP session is only instantiated by peers that support the presented RTP profile as it implements the exchange of repair packets. The header of repair packets is shown in Figure 2. The $n-k$ repair packets are generated by an optimal and systematic *C(n,k)* packet-level erasure code over k media packets.

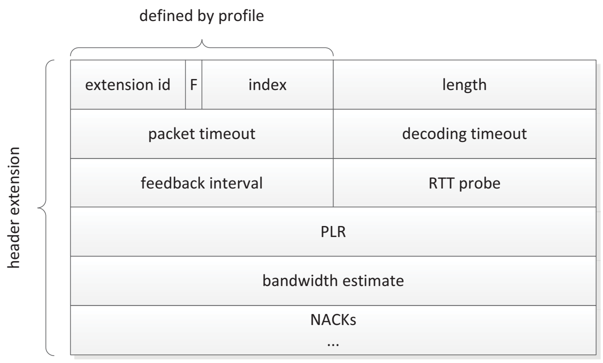

Figure 1. Media packet header extension.

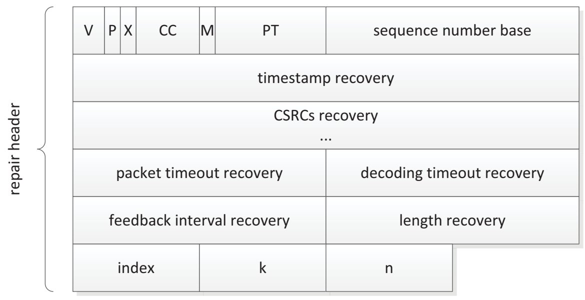

Figure 2. Profile-specific header parts for repair packets.

Similarly to RTP/FEC, repair packets contain recovery fields for the RTP header and the header extension of media packets. Network state feedback and NACKs are not included in the set of recovery fields as this information might already be outdated at the time the media packet is restored. The repair packet header additionally contains the following fields:

- The *sequence number base* states the sequence number of the first media packet in the associated coding block.
- The *index* field determines the position of the current packet in the coding block.
- *k* and *n* configure the block-erasure code.

B. Error Control

The protocol extension includes an adaptive HEC scheme that combines systematic block coding with NACK-based packet re-transmission. A subset of k arbitrary packets out of a block of k media packets and $n-k$ repair packets is sufficient to recover the k media packets. Repair packets might be sent proactively as well as in response to a NACK depending on the measured network state. A repair packet schedule $N_p[c]$, $0 \leq c \leq N_C$ specifies the number of repair packets to be sent in each of the N_C repair cycles. $N_p[0]$ denotes the number of repair packets that are sent proactively. The workflow of the coding scheme is depicted in Figure 3.

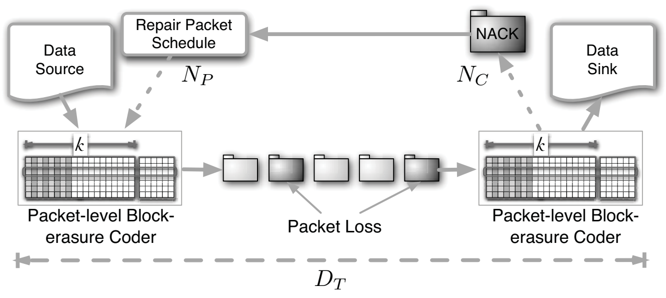

Figure 3. Hybrid error control of the predictably reliable RTP extension.

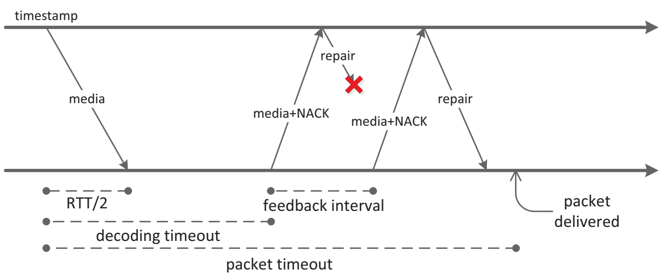

Figure 4. Timing model for delay-constrained error control.

C. Timing Model

In order to perform delay-constrained end-to-end error control, the reactive repair cycles are triggered at certain dynamically estimated timers. The schedule of receiver feedback is determined by two events: First, the sender must define the instant of time at which it expects feedback for the first reactive repair cycle. This date is communicated to the receiver as the *decoding timeout*. Second, after sending a loss notification, the receiver has to wait for the potential arrival of additional repair packets before it repeats the loss notification. Therefore, it determines an adequate *feedback interval*. An incoming loss notification is answered immediately by the sender. The block-internal time measurement starts with the first packet of the block entering the protocol stack. All timers are measured relatively to this instant of time. The timers ensure that the first packet of the block is delivered within the target end-to-end time constraint D_{Target}, which is defined by the application. The overall timing model includes the following components and is presented in Figure 4:

Decoding Timeout: The sender cannot answer to feedback before the successful completion of the block encoding operation. Moreover, the receiver does not know whether repair packets are already in transit at the time it observes the packet loss. Therefore, the transmission of all k source packets as well as the $N_P[0]$ proactive repair packets must be completed before the receiver decides whether additional redundancy is required for the decoding process.

Let T_S be the current source packet interval. Under the current source rate R_S and the source packet's payload length L_D, the packet interval at the sender is obtained by

$$T_S = \frac{L_D}{R_S}. \tag{1}$$

Consequently, the collection of k source packets causes a coding delay of

$$D_C = k \cdot T_s. \tag{2}$$

In an unsaturated network, i. e. if R_S does not exceed the available bandwidth, T_S is assumed to be greater than the transmission delay of the source packets. The coding process is implemented such that it does not delay the source packets at the sender. Therefore, the k source packets are assumed to be available at the receiver no later than

$$D_C + \frac{RTT + D_{RS}}{2} \tag{3}$$

time units after insertion of the first packet's first byte, where D_{RS} accounts for response latencies at the sender and the receiver. The repair packets are available at the receiver after a multiple of the parity transmission delay D_{PTx} under the assumption that the parity information at the sender is calculated within the response delay D_{RS}. Assume further that the receiver introduces an additional delay D_{PL} in order to detect packet losses in the transmitted block. The overall forward error coding delay D_{FEC} is thus obtained by

$$D_{FEC} = D_C + N_P[0] \cdot D_{PTx} + \frac{RTT + D_{RS}}{2} + D_{PL}. \tag{4}$$

Let the margin D_{FEC} be defined in order to compensate for the delay caused by block coding and original transmission. Let $t_{w,0}$ be the date where the first byte of the first source packet is available at the sender. The loss notification that activates the first repair cycle needs to be sent at time

$$t_{FB}[0] = t_{w,0} + D_{FEC}. \tag{5}$$

The value is calculated at the sender and communicated in the *decoding timeout* header of each source frame. The header field carries $t_{FB}[0]$ as an offset to the packet's *timestamp* field denoting the time t_s at which the packet is sent to the network medium, i. e. *decoding timeout* $= t_{FB}[0] - t_s = D_{FEC} - (t_s - t_{w,0})$.

Feedback Interval: After initiation of the reactive repair process by sending the first feedback at $t_{FB}[0]$, the receiver sends additional loss notifications periodically. The period of the receiver feedback is based on two observations: the network path's RTT as well as the transmission delay D_{PTx} of the repair packets sent. A delay margin D_{RS} additionally accounts for the response delay on the communicating hosts. Hence, if $N_P[c]$ repair packets are sent in cycle c, the whole request and repair cycle lasts for a request delay $D_{REQ}[c]$ of

$$D_{REQ}[c] = RTT + N_P[c] \cdot D_{PTx} + D_{RS}, \tag{6}$$

where $0 < c \leq N_C$.

Assume that the repair packets scheduled for repair cycle c arrive at the receiver with a request delay $D_{REQ}[c]$ after sending the loss notification for cycle c. Recall that the reactive repair process starts at time $t_{FB}[0]$. Consequently, the feedback for repair cycle $c+1$ is sent at

$$t_{FB}[c+1] = t_{FB}[0] + \sum_{i=1}^{c} D_{REQ}[c], \tag{7}$$

where $0 < c < N_C$. If the receiver cannot recover the coding block at time $t_{FB}[c+1]$, it repeats the loss notification for the block. This process lasts until either a sufficient number of repair packets is received or the remaining time budget is insufficient for the completion of another repair cycle. The overall time budget for each source packet is determined by the *packet timeout* header D_{TO} depending on the delivery delay constraint D_{Target}. If a packet enters the protocol stack at time t_w and is sent at t_s, the packet timeout D_{TO} is calculated as

$$D_{TO} = D_{Target} - (t_s - t_w). \tag{8}$$

Each source packet must be ready to leave the receiver's stack at the delivery deadline, which is $t_{w,0} + D_{Target}$ for each packet in the block. Depending on this date, a feedback deadline exists for the receiver after which additional repair packets cannot be received before the delivery deadline. The receiver must not send additional feedback for cycle $c+1$ if

$$t_{FB}[c+1] > t_{s,0} + D_{TO,0} - D_{REQ}[c+1], \tag{9}$$

where $t_{s,0}$ is the *timestamp* field and $D_{TO,0}$ is the *packet timeout* field of the coding block's first packet. In order to simplify the parameter signaling between sender and receiver, a constant feedback interval $D_{FB} = D_{REQ}[c]$, $0 < c \leq N_C$ is assigned to each repair cycle. The sender communicates D_{FB} in the header extension of each media packet via the *feedback interval*.

D. Protocol Workflow

The combination of block-erasure coding and reactive error repair requires the protocol stack to implement a highly concurrent architecture. The real-time media packet stream must be transmitted and received continuously with priority over error control operations, i. e. the block-erasure coding, loss detection, feedback scheduling and response to feedback. Therefore, it is required to parallelize the execution of the different tasks. In addition, the repair cycles of several adjacent blocks are usually overlapping in order to ensure a continuous flow of media packets. Figure 5 depicts the end-to-end workflow of the presented protocol extension. A bi-directional application implements both sending and receiving part of the protocol architecture.

Sending Part: Media packets are sent to the network right after insertion into the protocol socket. The protocol stack appends feedback about lost packets and the network state to the media packets in the backward RTP channel. At the same time the media packets are stored in a queue for further processing along with the individual coding parameters that are specified for the related coding block. For every k queued media packets the $n-k$ repair packets of the corresponding coding block are created and a task for sending the $N_P[0]$ proactive repair packets is scheduled in a task list. The task list stores information about all repair packets that are scheduled to be sent, including the reactive repair packets that are triggered from the NACKs of the backward channel.

The network state observed by the receiving part is analyzed from the received feedback. The adaption algorithm of the protocol's error control derives optimal coding parameters

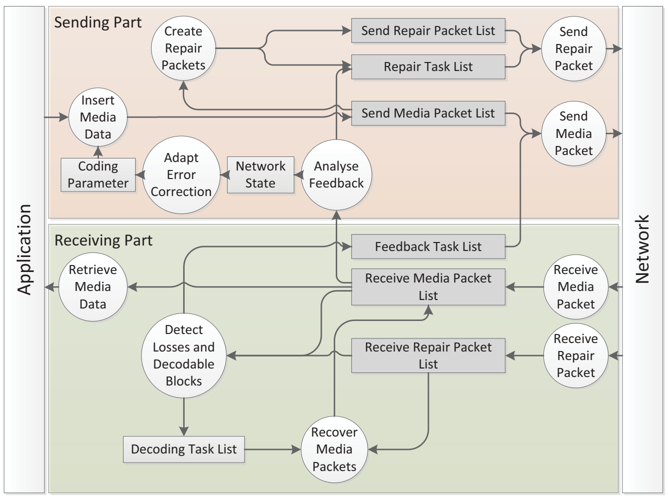

Figure 5. Workflow of sending (top) and receiving (bottom) part of the proposed RTP extension.

under the current network conditions. The sender adapts by continually adjusting the amount of redundancy that is added to the media stream. The redundancy depends on the protocol parameters k and N_P that are optimized based on the evaluation of the protocol performance model. Outdated media packets and corresponding redundancy packets are removed from the queue as soon as the target delay D_{Target} has expired.

Receiving Part: Incoming media and repair packets are stored in separate packet queues. The receiving packet queues provide a data structure for the re-sequencing of packets that are received out of order as well as for the elimination of duplicates. Further, this queue supports error detection and error correction at the receiver. If any NACKs from the backward channel are included in the media packets, they are extracted and converted into repair tasks (see *Sending Part*). Media packets in the receiving media packet queue are released to the application as soon as their packet timeout D_{TO} has expired. In case sequence number gaps are detected in the receiving packet queue, one of the following actions is required depending on the availability of repair packets: If a sufficient number of repair packets is available at the receiving part, the corresponding block is decodable and a decoding task is scheduled in the respective task list. A decoding task comprises gathering of the media and repair packets belonging to the specified coding block, recovery of the lost media packets and insertion at the right position in the receiving media packet queue. In the alternative case the lost packets cannot be recovered with the currently available media and repair packets. In this case the next feedback cycle is scheduled via a feedback task in the feedback list (see t_{FB}) and a feedback header is appended to a media packet in the backward channel as soon as it is due.

III. Performance Evaluation

In order to demonstrate the efficiency and the reliability of the presented RTP extension we evaluate the performance of the integrated adaptive error control scheme under typical delay requirements of interactive communications services. The protocol minimizes the error coding and the messaging overhead under strict target delay constraints via instantaneous adaptivity to the dynamically changing network conditions. Our performance evaluation concludes measurements under different network conditions with various packet loss rates and round trip delays.

A. Experimental Setup

The experimental setup comprises a Linux and a Windows platform which both are connected via Ethernet. We simulate a high quality video communications service with media source rates between $2\,Mbps$ and $8\,Mbps$ and with delay requirements between $150\,ms$ and $350\,ms$. In order to fulfills these application layer end-to-end delay requirements, the protocol's target delay D_{Target} is set to the respective values. In our experiments the Linux machine is acting as a sender while the Windows machine is acting as the receiver. This choice is arbitrary and does not influence the results. On the Linux machine an outgoing loss rate and a transmission delay are configured with the Netem[2] network emulator in order to set the evaluated scenario. For all test cases the size of the media packet payload is 1316 bytes and the target residual error rate after error correction is configured to be less than 10^{-5}. During each experiment 10^5 media packets are sent.

B. Results

Table I presents the measurement results. In general the residual error rate after the adaptive error control is below the target of 10^{-5}. Intuitively, the overhead in form of repair packets must be adjusted in order to compensate for the lost media packets at any time. In contrast to an FEC scheme with a constant, conservative configuration, however, this overhead dynamically follows the network's packet loss rate in the predictably reliable protocol extension due to the instantaneous performance modeling. Under a shorter time budget the protocol automatically applies error control with a stronger proactive component and few or no reactive repair cycles. As a result of this configuration, more repair packets are sent in advance.

A shortcoming of packet repetition schemes based on RTCP feedback consists in RTCP's strict feedback rules. As a part of our RTP extension, we specify the transport of feedback in the backward RTP channel of a bidirectional communications service. This modification allows for a higher feedback frequency without violating RTCP rules. The functionality of feedback is manyfold. It is required to signal channel conditions and to request repair packets. In our experiments the feedback consumes less than 1% of the session bandwidth. It varies between $2.5\,kbps$ for a low loss rate and a small target delay and $25\,kbps$ for the highest loss rate and the largest target delay.

[2]http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

Table I
ERROR CONTROL PERFORMANCE MEASUREMENTS

Network Conditions	D_{Target}	Media Source Rate	Sent Repair Packets	Residual PLR
PLR: $5\,\%$ Delay: $50\,ms$	$350\,ms$	$4\,Mbps$	$9.31\,\%$	0
	$250\,ms$	$4\,Mbps$	$10.21\,\%$	0
	$150\,ms$	$4\,Mbps$	$16.83\,\%$	$\approx 5 \cdot 10^{-6}$
PLR: $1\,\%$ Delay: $50\,ms$	$350\,ms$	$4\,Mbps$	$1.86\,\%$	$\approx 9 \cdot 10^{-6}$
	$250\,ms$	$4\,Mbps$	$2\,\%$	0
	$150\,ms$	$4\,Mbps$	$5.78\,\%$	0
	$350\,ms$	$8\,Mbps$	$1.96\,\%$	0
	$250\,ms$	$8\,Mbps$	$2.17\,\%$	0
	$150\,ms$	$8\,Mbps$	$5.61\,\%$	0
PLR: $1\,\%$ Delay: $100\,ms$	$250\,ms$	$8\,Mbps$	$5.57\,\%$	0
	$250\,ms$	$4\,Mbps$	$5.67\,\%$	0
	$250\,ms$	$2\,Mbps$	$6.3\,\%$	0

IV. CONCLUSION

In this paper we propose an RTP profile that enables adaptive error control for interactive real-time communication. It ensures backward compatibility to the baseline RTP standard. In contrast to available RTP profiles with partial reliability, it implements predictable reliability, which provides control over the residual packet loss rate. As a result of instantaneous and analytical adaptation of the error control, the coding and messaging overhead of the protocol is minimized. A specific workflow and a detailed timing model ensure the performance of the error control under the low delay requirements of interactive communications services. The detailed network state feedback of the extension prepares the protocol for various types of equation-based congestion control.

REFERENCES

[1] H. Alvestrand. Overview: Real Time Protocols for Browser-based Applications. Internet Draft (Work in Progress), February 2013.

[2] M. Gorius. *Adaptive delay-constrained internet media transport*. PhD thesis, Saarland University, December 2012.

[3] M. Gorius, J. Miroll, M. Karl, and T. Herfet. Predictable reliability and packet loss domain separation for IP media delivery. In *Proceedings of the IEEE International Conference on Communications (ICC) 2011*, pages 1–6, June 2011.

[4] R. Jesup. Congestion Control Requirements for RMCAT. Internet Draft (Work in Progress), February 2013.

[5] A. Li. RTP Payload Format for Generic Forward Error Correction. RFC 5109 (Proposed Standard), December 2007.

[6] H. Lundin, S. Holmer, and H. Alvestrand. A Google Congestion Control Algorithm for Real-Time Communication. Internet Draft (Work in Progress), February 2013.

[7] P. O'Hanlon and K. Carlberg. Congestion Control Algorithm for Lower Latency and Lower Loss Media Transport. Internet Draft (Work in Progress), April 2013.

[8] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). RFC 4585 (Proposed Standard), July 2006.

[9] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

[10] M. Welzl, S. Islam, and S. Gjessing. Coupled Congestion Control for RTP Media. Internet Draft (Work in Progress), June 2013.