Congestion Control using FEC for Conversational Multimedia Communication

Marcin Nagy
Aalto University
marcin.nagy@aalto.fi

Varun Singh
Aalto University
varun.singh@aalto.fi

Jörg Ott
Aalto University
jorg.ott@aalto.fi

Lars Eggert
NetApp
lars@netapp.com

5 October 2012

Abstract

In this paper, we propose a new rate control algorithm for conversational multimedia flows. In our approach, along with Real-time Transport Protocol (RTP) media packets, we propose sending redundant packets to probe for available bandwidth. These redundant packets are Forward Error Correction (FEC) encoded RTP packets. A straightforward interpretation is that if no losses occur, the sender can increase the sending rate to include the FEC bit rate, and in the case of losses due to congestion the redundant packets help in recovering the lost packets. We also show that by varying the FEC bit rate, the sender is able to conservatively or aggressively probe for available bandwidth. We evaluate our FEC-based Rate Adaptation (FBRA) algorithm in a network simulator and in the real-world and compare it to other congestion control algorithms.

1 Introduction

The development of Web Real-Time Communication (WebRTC) and telepresence systems is going to encourage wide-scale deployment of video communication on the Internet. The main reason is the shift from desktop or native real-time communication (RTC) applications (e.g., Skype, Google Talk, Yahoo and MSN messenger) to RTC-based web browsers or web applications. Currently, each web application implements their RTC stack as a plugin, which the user downloads. With WebRTC the multimedia stack is built into the web-browser internals and the developers need to just use the appropriate HTML5 API.

With the expected increase in multimedia traffic, congestion control is a re-emergent problem. These flows are subject to the fluctuations in path properties, such as packet loss, queuing delay, path changes, etc. Moreover, buffer bloat [1] and drop-tail routers can cause delay and bursty loss, which affects the user experience. Unlike in elastic applications, there are normally bit rate constraints on codecs, i.e., the codec encoding rates have a limited range of adaptation, and can only pick a few rates between these limits. Moreover, varying the encoding rate too often or in large steps degrades video quality [2]. Conversational multimedia differs from streaming multimedia because the former imposes a strict delay on end-to-end packet delivery. Consequently, it affects the size of the playout buffer, which for the conversational multimedia flows is at least an order of magnitude smaller than for streaming.

To tackle congestion control, the IETF has chartered a new working group, RMCAT[1], to standardize congestion-control for real-time communication, which is expected to be a multi-year process [3]. Therefore, [4] proposes minimum circuit breaker conditions under which a conversational multimedia flow should terminate its session. While this is not enough to perform congestion control, it reduces the effect of an unadaptive multimedia flow on the network [5].

Conversational multimedia systems use Forward Error Correction (FEC) and Packet Loss Indication (PLI) to protect against packet loss [6–8], i.e., the endpoint trades-off part of the sending rate for redundant packets or retransmissions to reduce the effect of losses on the user experience. An endpoint prefers to use FEC in networks where retransmissions would arrive later than the playback time of the packet and the loss rate is deterministic. Figure 1 shows the applicability of different error-resilience schemes based on network latency and packet loss [6]. If the loss rate differs significantly over several FEC intervals from the expected rate, the endpoint needs to adapt the FEC rate. To summarize, an RTC endpoint needs to adapts the sending rate to best fit the changing network capacity and the amount of FEC (or FEC interval) to best fit the observed network loss rate.

[1] http://tools.ietf.org/wg/rmcat/

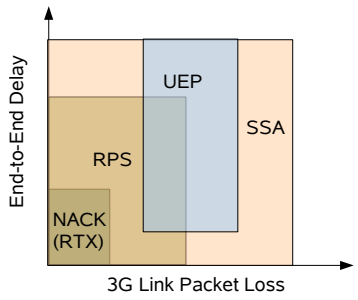


Figure 1: shows the applicability of an error-resilience scheme based on the network delay and packet loss [6]. UEP is Unequal level of protection implemented by FEC, SSA is slice size adaptation, RPS is reference picture selection, NACK are retransmissions.

Therefore, in this paper, we investigate the use of redundant packets not only for error-concealment but also as a probing mechanism for congestion control (ramping up the sending rate). Other error-resilience methods like, retransmissions (such as, ARQ [6]), sending duplicate packets [9] (reduces network efficiency), error-concealment at the sender and receiver [8] are ignored.

By choosing a high FEC rate, an endpoint aggressively probes for available capacity and conversely by choosing a low FEC rate, the endpoint is conservative in probing for additional capacity. If during probing for additional capacity, a packet is lost due to congestion, the receiver may be able to recover it from the FEC packet (i.e., if the FEC arrives in time for decoding). If no packet is lost, the sender can increase the media encoding rate to include the FEC rate. This method can be especially useful when the sending rate is close to the bottleneck link rate, by choosing an appropriate FEC rate the endpoint can probe for available capacity without affecting the baseline media stream.

To verify that FEC can be applied to multimedia congestion control (for WebRTC), we propose a new RTP rate adaptation algorithm, termed *FEC Based Rate Adaptation*–FBRA (see Section 4). We evaluate its applicability using ns-2 [10] in various simulation scenarios and in a real-world testbed (Section 5). Furthermore, to better understand the concept, we have developed a simplified state machine of the algorithm (Section 3). Finally, we present also a detailed description of the design of the *Adaptive Multimedia Subsystem* (Section 6).

2 Related Work

The Real-time Transport Protocol (RTP) [11] is used to deliver conversational multimedia flows and is favored over TCP for media delivery due to the very stringent timing requirements of multimedia [12]. RTP carries the media packets while the RTCP reports carry media playout and sender-to-receiver path statistics, such as, jitter, RTT, loss rate, etc. The media playout information can be used to synchronize the audio and video streams at the receiver and the path statistics are used by the sender to monitor the session.

Standard RTP limits the reporting interval to a minimum of $5 \pm 2.5s$ to avoid frequent reporting. Due to the long reporting interval, the end-to-end path statistics become too coarse-grained to be applicable for congestion control. However, RFC4585 [13] removes this minimum reporting interval constraint and endpoints can use up to 5% of the media rate for RTCP. With the smaller reporting interval, the congestion control algorithm can expect feedback packets on a per-packet, per-frame or per-RTT basis [14]. Using RTCP Extended Reports (XR) [15], an endpoint can report other path heuristics, such as discarded packets, bursts and gaps of losses, playout delay, packet delay variation, among others.

Several sender-driven congestion control algorithms have been proposed over the years. Most prominent is the TCP Friendly Rate Control (TFRC) [16], which can be implemented using the information contained in standard RTCP reports (e.g., RTT and loss measurements). However, TFRC requires feedback on a per-packet basis [17], which can produce an increase and decrease in the media rate (sawtooth) in a very short interval of time [18, 19]. RAP [20] uses a token bucket approach to additively increase and multiplicatively decrease the rate (AIMD), but as the media rate reaches the bottleneck rate the encoding rate in this case as well exhibits a sawtooth-type of behavior. Due to the impact on perceived media quality, any algorithm that consistently produces a sawtooth-type of behavior is not well suited for conversational multimedia communication [21, 22].

Instead of just relying on RTT and loss for congestion control, Garudadri *et al.* [23] also use the receiver playout buffer to detect under and overuse and schedule RTCP feedback packets every 200-380ms to have timely feedback. Singh *et al.* [24] use a combination of congestion indicators: frame inter-arrival time, playout buffer size for congestion control. Zhu *et al.* [25] use ECN and loss rate to get an accurate estimate of losses for congestion control. O'Hanlon *et al.* [26] use a delay-based estimate when competing with similar traffic and use a windowed-approach when competing with TCP-type cross traffic,

they switch modes by using a threshold on the observed end-to-end delay, their idea is similar to the one discussed in [27]. Holmer *et al.* [28] proposes a Receive-side Real-time Congestion Control (RRTCC) algorithm, which uses the variation in frame inter-arrival time to detect link under and overuse. The new media rate is calculated at $1s$ intervals by the receiver and signaled to the sender in a Temporary Maximum Media Stream Bit Rate Request (TMMBR) message [29]. RRTCC also does not react to losses less than 2%, instead increases the rate until 10% losses are observed. Recent analysis of RRTCC shows that it performs poorly when competing with cross-traffic [30, 31].

Most of the above literature for congestion-control of conversational multimedia considers using error resilience and congestion control separately. Zhu *et al.* [32, 33] propose using Unequal Loss Protection (ULP) for both congestion control and error-resilience. Firstly, they estimate the available rate using a variant of TFRC, called Multimedia Streaming TCP-friendly protocol (MSTFP) [34]. Secondly, they take packet loss and historical sending rate to smooth out the encoding rate. Lastly, they apply FEC while performing congestion control and their results show a significant increase in Peak Signal-to-Noise Ratio (PSNR). MSTFP does not use RTP/RTCP, applies it to streaming video and acknowledges each packet for calculating the TFRC estimate. While our proposal applies to conversational video with tight delay requirements.

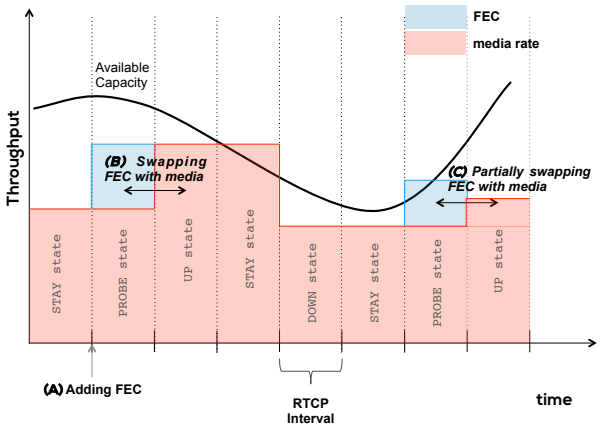


Figure 2: shows the concept of using FEC for rate control. When congestion cues indicate no congestion, the FEC stream is enabled to probe for available bandwidth. When no losses are reported in the next RTCP report, the media bit rate is increased. When congestion is observed, the congestion control algorithm reduces the media rate.

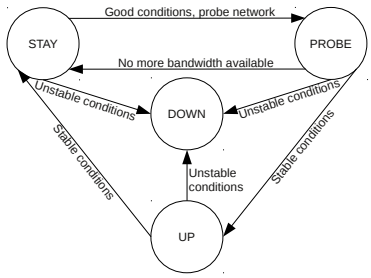


Figure 3: State machine of a congestion control algorithm enabling FEC. It shows that after enabling FEC three conditions may occur. 1) No more bandwidth is available, the sender should keep the current rate, 2) Stable conditions are detected, the sender should increase the rate and disable FEC, and 3) Unstable conditions are detected, and the sender should reduce the rate to the goodput.

3 Concept: Using FEC for Congestion Control

FEC is one method of error protection that improves flow reliability by adding redundant data to the primary flow which is used by the receiver to recover parts that have been lost due to either congestion, or bit-errors. The rate control algorithm on the other hand aims at providing the best possible network path utilization, but risks overestimating the available end-to-end capacity that may lead to congestion induced losses.

The main idea behind using FEC for rate control is to enable FEC alongside the media stream and use it to probe the path for available bandwidth. If the path conditions are good and stable after the FEC stream is switched on, the media encoding rate is increased by the amount of the FEC stream rate and the FEC stream is disabled (see figure 2). The main advantage of this approach is that the applied rate control algorithm can be more adventurous in probing for available capacity, as improved reliability compensates for possible errors resulting from link overuse.

Figure 3 illustrates the state machine of a congestion control algorithm incorporating FEC for probing. The state machine includes 4 states: ***STAY***, ***PROBE***, ***UP***, and ***DOWN***. The rate adaptation algorithm decides based on the congestion cues (such as, RTT, loss/discard rate, jitter, packet delay variation, ECN, etc.) to stay in the current state, or transit to another. The state machine does not specify the conditions for the transition between states, but only gives a very generic description of the path conditions and leaves the interpretation to the underlying con-

gestion control algorithm.

The primary consideration for probing using FEC is *"How much FEC should be introduced alongside the media stream?"* If the endpoint uses a higher FEC rate, it has better protection against losses, irrespective of what FEC scheme it is using. Moreover, the ramp-up is quicker but at the risk of overloading the path and causing congestion. If the endpoint uses a lower FEC rate, it has weaker protection against losses and also a potentially slower ramp-up. Therefore, the sender should observe the congestion cues to make its decision. If the congestion cues indicate that it is operating close to the bottleneck link, it should use lower FEC rate; however, if the cues indicate that it is underutilizing the link it should use a higher FEC rate. Hence, an important aspect of the rate adaptation algorithm is the ability to find the correct FEC rate for the current network conditions. Another aspect to consider is the FEC scheme. It determines the amount of redundancy injected into the network. An application may employ FEC at the packet-level, frame-level or use an unequal level of protection (ULP).

4 FEC-Based Rate-control Algorithm (FBRA)

In this section, we describe our proposed congestion control algorithm and the RTP/RTCP extensions it uses. We also describe the conditions under which to enable FEC.

4.1 Using RTP/RTCP Extensions

Our algorithm uses a short RTCP reporting interval and our experiments show that the interval need not be shorter than $2 \times RTT$. In exigent circumstances, such as the receiver detects that the playout buffer is about to underflow due to late arrival of packets[2] then the receiver may send the RTCP feedback early, however, an endpoint can only send an early feedback once every other reporting intervals [13].

Apart from the congestion cues reported in the standard RTCP Receiver Report (RR), such as jitter, RTT, and loss rate, we additionally use Run-length encoded (RLE) lost [15], and RLE discarded [35] packets. Using the RLE lost and discarded packets[3] the sender can correlate when exactly the loss and discard events took place in the reporting interval. If these events occurred earlier in the reporting interval, the sender may ignore them as transient and keep the same sending rate. Conversely, if the events occurred later in the interval (more recent) then the sender would calculate the exact goodput[4] and use that as the sending rate. Furthermore, using the RLE lost and discarded packet information the receiver may be able to distinguish between bit-error losses and congestion losses [24].

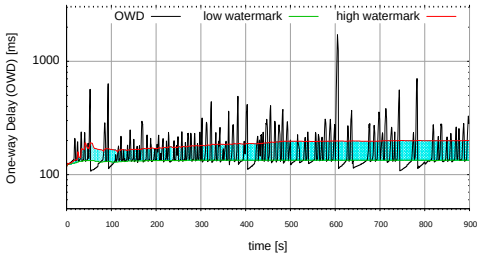


Figure 4: The plot shows the variation in measure OWD value (dark line) during a video call, the low and high watermark represent the 40^{th} and 80^{th} percentile. The FBRA algorithm uses the α and β thresholds to validate, if the recently measured OWD value falls in the region of operation (shaded area).

RTT, as a congestion cue, has one fundamental issue, namely it assumes that the network paths in both directions are symmetric and that the congestion upstream is similar to the congestion downstream, which may not be always true [36]. As a result, we use the technique defined in [36] to precisely calculate the one-way delay (OWD) at the receiver without the need for clock synchronization and the receiving endpoint reports the observed OWD in the reporting interval using the extension defined in [37]. Increase in the one-way delay may indicate a queuing delay in the network, which is an early sign of congestion. To calculate the uncongested OWD, the sender only collects the reported OWD from the RTCP RRs that do not report any losses or discards into a data structure, $OWD_{history}$. The sender then uses the $OWD_{history}$ to determine the optimum OWD.

As most of the $OWD_{history}$ values are close to the "ideal" OWD value, we use the 40^{th} percentile[5] to determine the threshold for increasing the sending rate (low watermark), i.e., if the current OWD measurement is smaller than the 40^{th} percentile value indicates link under-utilization.

Similarly, we use the 80^{th} percentile, (which is at the higher end of the distribution) to set the threshold for de-

[2]Typically underflow may occur due to routing updates or queuing delay at an intermediate router.

[3]Packets that arrive too late to be displayed are discarded by the receiver.

[4]using the history of sent, lost and discarded packets in the last reporting interval [19].

[5]in our experiments, we find that the 40^{th} percentile value provides better tolerance than the median.

creasing the rate (high watermark), i.e., if the most recent OWD measurement is larger than the 80^{th} percentile value indicates the onset of congestion (see Figure 4). This method is very similar to the one used in [19], which uses RTT measurements instead of OWD.

$$Correlated_{lowOWD} = \frac{current_{OWD}}{40^{th} percentile(OWD_{history})}$$

$$Correlated_{highOWD} = \frac{current_{OWD}}{80^{th} percentile(OWD_{history})}$$

The FBRA algorithm uses the correlated OWD by checking if its value exceeds thresholds: α and β. α and β thresholds values must be within [1;2] interval, and have been derived empirically in a series of experiments. Because these values depend on the state of the congestion control algorithm, the exact thresholds are described in more detail in Section 4.3.

4.2 Size of $FEC_{interval}$

In this paper, we use FEC for both rate adaptation and for the error protection because we can adapt the rate without the fear of creating packet losses that impair QoE. We use a basic per packet parity FEC scheme because it gives 1 redundant packet for every *N* RTP packets. By varying the number of RTP packets encoded by a single FEC packet (known as the $FEC_{interval}$) the congestion control algorithm can vary the ramp-up rate, i.e., to increase redundancy the congestion control algorithm only needs to reduce the $FEC_{interval}$. Therefore, for the rest of the paper, we have ignored the use of more complex FEC schemes and leave it for further study.

We have limited the $FEC_{interval}$ to encode between 2 and 14 RTP packets. The minimum $FEC_{interval} = 2$ is the lowest possible and creates maximum redundancy and high FEC rate, whereas, the maximum $FEC_{interval} = 14$ creates low redundancy and a low FEC rate. In our experiment, we found that a $FEC_{interval} > 14$ had negligible impact on both congestion control (because the additional FEC rate is too small) and for error protection (because the FEC packet is generated too late to help in decoding the lost packet, the playout buffer of an interactive flow is very small).

The $FEC_{interval}$ is calculated based on the following observations. When the RTP media rate is low, it is assumed that there is a lot of available end-to-end capacity for the media stream, and the $FEC_{interval}$ is set low. This allows the sender to quickly ramp-up the sending rate, and if the link is overused it also provides higher protection against possible losses. Conversely, when the RTP media rate is high, it is assumed that the sender is approaching the bottleneck link capacity and there may not be much more bandwidth left for the media stream, and the $FEC_{interval}$ is set to a high value. This allows the sender to ramp-up slowly and avoid overshooting the bottleneck link capacity. To determine if the current media rate is low or $high$, the sender keeps a history of all the rates (goodput, sending rate, combined FEC and RTP sending rate) recorded in the last two seconds, and compares the current rate with the highest recorded value in the set and with the initial goodput.

Sub-algorithm 1a DOWN state function

```
1:  function STATE DOWN
2:      if Recent Losses ∨ Discards then
3:          if PreviousState = DOWN then
4:              NewState ← STAY
5:          else
6:              if Discards ∧ No losses then
7:                  Undershoot without disabling rate control
8:              else
9:                  Undershoot and disable rate control
10:             end if
11:             NewState ← DOWN
12:         end if
13:     else if Corr_highOWD > α_undershoot then                ▷ α_undershoot = 2.0
14:         Undershoot and disable rate control
15:         NewState ← DOWN
16:     else
17:         NewState ← STAY
18:     end if
19:     Disable FEC
20: end function
```

4.3 FBRA: Algorithm

The FBRA algorithm has the following states: *STAY*, *PROBE*, *UP*, *DOWN*. The state names match exactly the ones described in Section 3. The algorithm changes state depending on the information received in the new RTCP RR and is constrained by the following state transition rules.

In the ***DOWN* state** the algorithm (see Sub-algorithm 1a), reduces the sending rate by executing the *undershooting procedure*. If no congestion is reported in the next RTCP interval the algorithm transits to the *STAY* state. However, if losses and discards still appear the algorithm stays in the *DOWN* state. In the edge case, when high OWD values are reported, the *DOWN* state is also kept. As the algorithm should not be very sensitive to high OWD values during congestion, the threshold for the edge case is set to $\alpha_{undershoot} = 2.0$.

In the ***STAY* state** the algorithm (see Sub-algorithm 1b), keeps the sending rate constant, and the FEC packets are not generated. The algorithm can remain in this state and not probe for additional capacity

Sub-algorithm 1b STAY state function

```
1: function STATE STAY
2:     if Losses then
3:         if Recent Losses then
4:             Undershoot and disable rate control
5:             NewState ← DOWN
6:         else
7:             NewState ← STAY
8:         end if
9:         Disable FEC
10:    else
11:        if Recent Discards then
12:            Undershoot, disable rate control and FEC
13:            NewState ← DOWN
14:        else
15:            if Corr_highOWD > α_stay then        ▷ α_stay = 1.1
16:                if PreviousState = STAY then
17:                    Undershoot and disable rate control
18:                    NewState ← DOWN
19:                else
20:                    NewState ← STAY
21:                end if
22:                Disable FEC
23:            else
24:                NewState ← PROBE
25:                EnableFEC
26:            end if
27:        end if
28:    end if
29: end function
```

Sub-algorithm 1c PROBE state function

```
1: function STATE PROBE
2:     if Recent Losses ∨ Recent Discards then
3:         Undershoot, disable rate control and FEC
4:         NewState ← DOWN
5:     else if Losses ∨ Discards then
6:         NewState ← STAY
7:         Disable FEC
8:     else
9:         if Corr_highOWD > α_down then    ▷ α_down ∈ [1.4; 1.6]
10:            Undershoot, disable rate control and FEC
11:            NewState ← DOWN
12:        else if Corr_highOWD > α_stay then        ▷ α_stay = 1.1
13:            NewState ← STAY
14:            Disable FEC
15:        else if Corr_lowOWD > β then              ▷ β = 1.2
16:            IncrementFECinterval
17:            NewState ← PROBE
18:        else
19:            NewState ← UP
20:            NewRate ← CurrentRate + FECRate
21:            Disable FEC
22:        end if
23:    end if
24: end function
```

Sub-algorithm 1d UP state function

```
1: function STATE UP
2:     if (RecentLosses ∨ Discards ∨ Corr_highOWD > α_down
   then                                    ▷ α_down ∈ [1.4; 1.6]
3:         Undershoot and disable rate control
4:         NewState ← DOWN
5:     else
6:         NewState ← STAY
7:         Disable FEC
8:     end if
9: end function
```

if the congestion cues indicate that the sender is operating very close to the bottleneck link capacity. Otherwise, it can transit to the *PROBE* state for probing the path for additional capacity or for error-resilience. In either case, the stream may benefit with stability or increased error-resilience. In order to switch to the *PROBE* state, the $Correlated_{highOWD}$ must be lower than $\alpha_{stay} = 1.1$. Direct transition to the *PROBE* state is not possible when the current sending rate is higher than 90% of the highest rate recorded in the last 2 seconds. In this case, the algorithm assumes that the current rate is operating close to the bottleneck link capacity, and it must make sure that current rate is stable by staying in the current state for one more RTCP report interval before probing further. Obviously, if congestion is detected, the algorithm goes from the *STAY* state to the *DOWN* state.

In the ***PROBE* state** the algorithm (See Sub-algorithm 1c) maintains the current sending rate, but sends FEC packets alongside. If the next RTCP report shows signs of congestion, the algorithm disables FEC and goes back to the *STAY* state and if further congestion is detected, it goes to the *DOWN* state. Otherwise it normally transitions to the *UP* state. When no losses and discards are observed, the state transition is based on the measured OWD. If the observed OWD is higher than the $Correlated_{highOWD}$, the sender assumes that congestion is severe and thus cuts the sending rate more drastically. Our experiments show that the best results are obtained when α_{down} parameter, which is the threshold for entering the *DOWN* state is between 1.4 and 1.6. Because in this state, it is desirable to find out if the link is underutilized, less sensitivity is needed. Thus, we choose $\alpha_{down} = 1.6$. Less sensitivity, increases possibility of getting discards, but appearance of them allows us to find out about the link limit. Transition back to the *STAY* state occurs for correlation values exceeding $\alpha_{stay} = 1.1$ (similarly to the *STAY* state). Furthermore, the algorithm may also decrease amount of the FEC rate if the OWD value is unexpectedly higher. This condition is checked by comparison the $Correlated_{lowOWD}$ value against the β parameter. If the OWD value is low enough the *UP* state is entered, thus β parameter should be close to the lower boundary of [1;2] interval (we use $\beta = 1.2$).

In the ***UP* state** the algorithm (See Sub-algorithm 1d) increases the sending rate by replacing the FEC rate with additional RTP media rate. If congestion is detected, the algorithm transits to the *DOWN* state, or else to maintain stability for one more reporting interval, it transits to the *STAY* state. Transition to the *DOWN* state happens when the $Correlated_{highOWD}$ exceeds α_{down} value. We use $\alpha_{down} = 1.4$, as in the *UP* state more sensitivity for early congestion indication is required.

The algorithm is also sensitive to the RTCP reporting interval duration. This means that if the sender receives an RTCP RR at an interval shorter than $1.5 \times RTT_{median}$, it assumes this is an early report and immediately transits to the *DOWN* state. Furthermore, If no RTCP report is received for $2s$, the sender halves the rate entering the *DOWN* state [4].

4.4 FBRA: Undershooting & Bounce-back Procedure

We define two additional procedures: the *undershooting* and the *bounce-back* procedure. The undershooting procedure attempts to reduce the overuse of the network caused by the stream. The stream sets the new sending rate to a value lower than the current goodput [19,28], i.e., the sender calculates the new sending rate by subtracting twice the difference between the current sending rate and the current goodput and taking 90% of the obtained value.

After undershooting, if the sender predicts that the congestion cues in the upcoming reports may still show signs of congestion due to the previous overuse, the sender may disable the rate adaptation for a brief period of time. This action prevents any further reduction in the sending rate, while waiting for the congestion cues to stabilize. The deactivation period is is a bit more than the *RTCP Interval*, i.e., the sender ignores the next RTCP report that arrives. After the deactivation period expires, the bounce-back procedure is executed.

In the bounce-back procedure, the sender examines the most recent RTCP report (the one received after the period expires), and if there are no signs of congestion, it increases the sending rate to 90% of the goodput stored during the undershooting. The algorithm in this case attempts to gradually bring the sending rate back to the goodput observed at the moment of undershooting. However, if the new RTCP RRs continues to report congestion, the FBRA once again enters the undershooting procedure and this time does not disable the congestion control.

5 Performance Evaluation

In this section, we evaluate the performance of our proposed algorithm, FBRA, RRTCC [28] and C-NADU [24] in ns-2 [10]. Our simulations comprise of the following scenarios: a single RTP flow on a variable link capacity, one or more RTP flows competing on a bottleneck link, and one or more RTP flows competing with other short TCP flows on a bottleneck link. This paper mainly focuses on the congestion control in the Internet environment, where bit-error losses are few, therefore, we use simple duplex-links with no link loss rate. For intermediate routers we set the queue length to 50 packets (ns-2 default) and drop-tail queuing strategy, i.e., the packet loss observed in the simulation results are due to link overuse. In all the scenarios, we further divide the simulations into three sub-scenarios with each sub-scenario using a different bottleneck link delay (50 ms, 100 ms, and 240 ms) [12, 38, 39]. For statistical relevance, each sub-scenario is simulated 30 times and the standard deviation is noted for each metric. The simulation scenarios and the corresponding evaluation parameters are based on those defined in the RTP evaluation framework [40].

Furthermore, as the simulations are performed at the packet/frame level, we decided not to differentiate between the different types of video frames (namely, I-, P-frames), but represent each frame as a packet of equal size corresponding to the instantaneous sending rate. However, the sender may fragment large frames (at high bit rate) to fit the MTU size ($= 1500$ bytes). The endpoints generate the frames at 30 FPS and all three congestion control algorithms use the same frame rate and packetization methodology. Startup rate is outside the scope of this paper and therefore both endpoints begin their session with an initial sending rate of $128kbps$, and restrict the minimum rate for each congestion control algorithm to $32kbps$. There is no restriction on the maximum rate but, the maximum allowed end-to-end packet delay ($delay_{max}$) for a packet is set to $400ms$ [38], packets arriving after this cut-off are discarded by the receiver without sending them to the playout buffer.

5.1 Metrics

Apart from the standard metrics—e.g., goodput, Peak signal-to-noise ratio (PSNR), Packet Loss Rate (PLR), Average Bandwidth Utilization (ABU)—for evaluating the impact of the congestion control algorithm, we propose three new metrics to evaluate the performance of using FEC for rate control.

FEC Rate-Control Correctness: FRCC specifies times the fraction of time the congestion control algorithm correctly uses FEC, i.e., the algorithm starts from the *STAY* state, enables FEC and returns to the *STAY* state without entering the *DOWN* state. Consequently, the decision is incorrect if the algorithm enters the *DOWN* state after enabling FEC. Figure 3 shows which state transitions are allowed and which are not. Formally, we define:

$$FRCC = \frac{count(FEC\ raises\ rate) + count(FEC\ keeps\ rate)}{count(FEC\ enabled)}$$

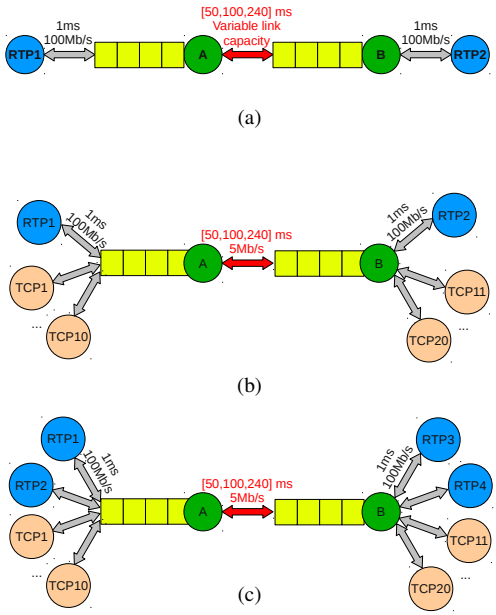


Figure 5: Shows the *ns-2* simulation topologies for (a) single RTP flow on a variable capacity link and (b) One or more RTP flows competing with multiple short TCP flows.

FEC Frame Recovery Efficiency: FFRE specifies the fraction of successfully recovered frames from all the lost frames that were protected by FEC. Formally:

$$FFRE = \frac{frames_{recovered}}{frames_{protected\ but\ lost} + frames_{recovered}}$$

TCP fair share: TFS specifies the ratio between the TCP flow throughput and the fair amount of bandwidth that should be granted to it. The ratio can be greater than 1, if TCP uses more than its fair share. Formally:

$$TFS = \frac{\left(\frac{TCP\ Throughput}{no.of\ TCP\ flows}\right)}{\left(\frac{Total\ Throughput}{no.\ of\ flows}\right)}$$

5.2 Single RTP flow on a Variable Capacity Link

Figure 5a illustrates this scenario topology, an RTP node is simulating a video conversation with another node connected through a constrained bottleneck link (*dumbbell* topology). The access links have a capacity of $100Mbps$ and $1ms$ delay, while the bottleneck link capacity varies between $100kbps$ and $256kbps$. In the scenario, we evaluate the reactivity and convergence of the congestion control algorithm to the available end-to-end capacity.

Table 1: Overall metrics for an RTP Flow on a Variable Capacity Link

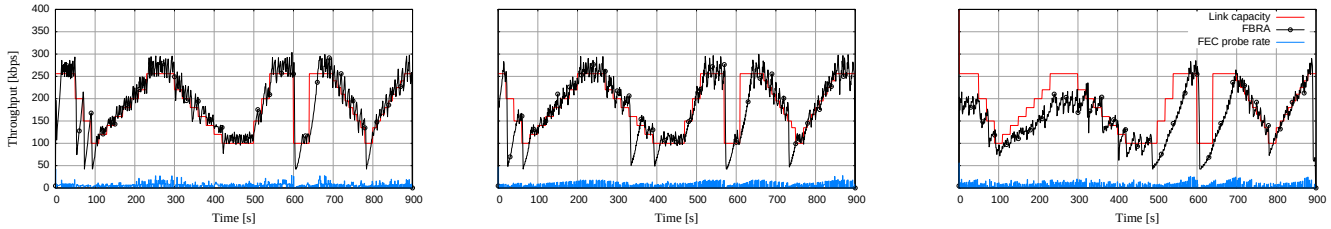
Delay	Metric	FBRA		RRTCC		C-NADU	
		avg.	σ	avg.	σ	avg.	σ
50ms	Goodput [kbps]	179.13	2.26	181.8	3.11	165.42	3.87
	Loss rate [%]	1.23	0.28	4.27	0.78	0.34	0.11
	No. of lost frames	441.43	82.37	1842	25.4	93.67	29.64
100ms	Goodput [kbps]	172.83	2.74	172.48	6.6	163.84	3.11
	Loss rate [%]	1.72	0.37	3.09	0.85	0.17	0.09
	No. of lost frames	562.83	103.44	740	42.82	46.4	22.94
240ms	Goodput [kbps]	144.89	8.35	169.22	5.68	153.52	6.81
	Loss rate [%]	2.82	0.89	2.98	0.55	0.19	0.07
	No. of lost frames	789.93	223.55	705.67	41.33	53.23	21.41

Our simulations show that RRTCC achieves the best goodput (169-180$kbps$) for the three bottleneck link delays (see Table 1), but has the worst loss rate (3-4$\%$). RRTCC is aggressive in its probing for available bandwidth and as per the algorithm defined in [28] does not react to losses up to 2$\%$. On the other hand, C-NADU achieves excellent reliability (0.15-0.5$\%$) results in all the cases, but has lower goodput (about 10-15$kbps$ lower than RRTCC). The two algorithms trade-off throughput for packet loss and vice-versa.

For the $50ms$ and $100ms$ bottleneck link delay, the goodput achieved by FBRA is comparable to RRTCC but with comparatively lower loss rates ($\approx$1.5$\%$). Figures 6(a)–(b) show that the FBRA can quite quickly bounce-back after undershooting. The figure also shows that the FEC probing rate increases when the FBRA ramps up and the FEC rate is low or disabled when the FBRA undershoots. However, FBRA is primarily a delay-based control algorithm and for the $240ms$ bottleneck delay, it observes that the packets are arriving very close to the $delay_{max}$ and is therefore, conservative in its probing for available bandwidth (this is observed by the low FEC rate in Figure 6c). FEC rate is about 10% of the media rate at about 15-20$kbps$ and the accuracy of using FEC for congestion control ($FRCC > 90\%$) is also very high in each case.

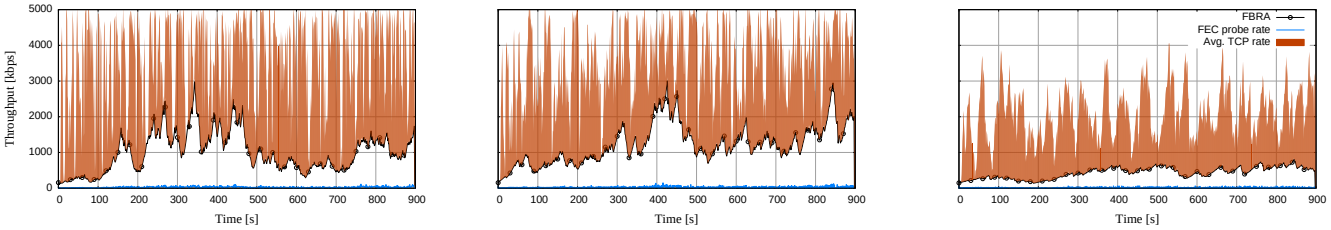
5.3 One RTP flow competing with multiple short TCP flows

In this simulation experiment, we evaluate the performance of the RTP flow when it competes with a number of TCP flows. Similar to the previous scenario, we use a dumbbell topology but instead of just a single RTP flow traversing the bottleneck link, it is shared with 10 short TCP connections (for e.g., having 10 tabs open in a browser). Each TCP flow is modeled as a sequence



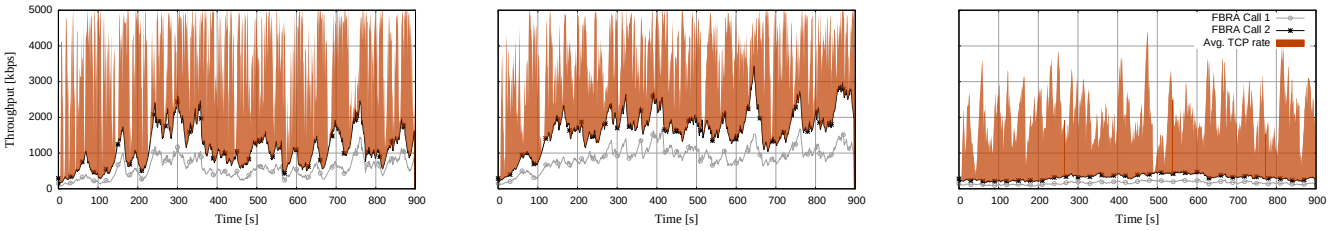
(a) OWD=50ms, FRCC=92.6% (b) OWD=100ms, FRCC=95.9% (c) OWD = 240ms, FRCC=90.1%

Figure 6: The plot shows the performance of a single RTP flow using FBRA in a varying link capacity scenario with different bottleneck delays. The plots also show the FEC probing rate. We observe that the FEC rate is low when the FBRA rate drops and FEC rate is high when the FBRA is ramping-up, but with $FRCC > 90\%$ in all the cases shows that the FEC probing was accurate. The bandwidth utilization in the high delay ($240ms$) scenario is low because the FBRA senses that the one-way delay is very close to the $delay_{max} (= 400ms)$ and is conservative in its bandwidth probing.



(a) OWD=50ms, FRCC=90.5% (b) OWD=100ms, FRCC=87.8% (c) OWD=240ms, FRCC=83.6%

Figure 7: The plot shows the performance of a single RTP flow using FBRA when competing with 10 short TCP flows on a common bottleneck link. The bottleneck link capacity is a constant $5Mbps$. To show the bottleneck link utilization, the FBRA rate and the Average TCP rate are stacked on top of each other. The FEC probing rate is plot independently to show that the FEC probing rate correlates with the FBRA sending rate. As before, the link utilization in the very high delay ($OWD = 240ms$) scenario is low because the FBRA sense that it is operating very close to the $delay_{max}$.



(a) OWD=50ms, FRCC=90.5% (b) OWD=100ms, FRCC=88% (c) OWD=240ms, FRCC=81.75%

Figure 8: The plot shows the performance of two RTP flows using FBRA competing with 10 short TCP flows on a bottleneck link with different delays. The sending rate for the two RTP flows and the average TCP throughput are stacked on top each other. The flows in all cases appear fair to one another, but as observed in the previous scenarios the FBRA is very conservative in probing for available bandwidth in the high delay scenario ($OWD = 240ms$).

of file downloads interleaved with idle periods (on-off traffic simulating webpage downloads). The sizes of the webpages are obtained from a uniform distribution between $100KB$ and $1.5MB$. Lengths of the idle periods

Table 2: Overall metrics for 1 RTP Flow competing with 10 TCP Flows

Delay	Metric	FBRA		RRTCC		C-NADU	
		avg.	σ	avg.	σ	avg.	σ
50ms	Goodput [kbps]	1044.24	122.9	3592. 9	279	2657.1	164
	Loss rate [%]	1.95	0.26	3.68	0.26	0.67	0.15
	No. of lost frames	821.63	106.5	5292	153	1418	134
	TCP flow throughput [kbps]	669.76	5.41	688.5	52.98	502.28	94.75
	TFS [%]	147.35	1.19	151.2	7.59	142.27	2.3
100ms	Goodput [kbps]	1219.83	210.11	3699.43	419.81	2968.84	59.38
	Loss rate [%]	1.54	0.17	4.05	0.19	0.64	0.09
	No. of lost frames	685.47	89.34	6210.82	261.12	1422	49.26
	TCP flow throughput [kbps]	491.34	4.17	323.13	110.99	515.29	68.73
	TFS [%]	108.1	0.92	71.09	0.21	113.37	0.73
240ms	Goodput [kbps]	504.82	82.18	3827.42	423.55	3028.22	78.55
	Loss rate [%]	0.4	0.05	4.82	0.17	0.36	0.05
	No. of lost frames	165.07	27.7	8547.93	526.8	597.24	123.96
	TCP flow throughput [kbps]	299.12	2.56	262.4	88.34	291.22	37.05
	TFS [%]	65.81	0.56	57.73	0.13	64.07	0.6

are drawn from an exponential distribution with the mean value of 10 seconds. Unlike the previous scenario, the bottleneck link has a constant available capacity of $5Mbps$ and the topology is illustrated in the Figure 5b.

In our simulations, the RRTCC produces the highest goodput (3.6-3.8$Mbps$), but with high variation (see standard deviation in Table 2). Additionally, the RTP flow experiences high packet loss (3.5-5%). C-NADU makes the opposite trade-off between throughput and loss rate and therefore, has lower goodput (2.6-3$Mbps$) and lower loss rate (0.3-0.7%).

Similar to the previous scenario, the FBRA algorithm performs very well in the 50ms and 100ms delay cases with goodput over $1Mbps$ and compared to RRTCC significantly lower loss rate (0.4-2.0%) and standard deviation of goodput results. In the 240ms bottleneck delay case the goodput falls to 505kbps because the FBRA being delay-based becomes conservative. Furthermore, we observe that at 240ms, FEC becomes more useful for error protection ($FFRE = 18\%$) than for congestion control ($FFRC = 83.6\%$). For other delays the FFRE is lower (11-12%) and FFRC is higher (87-91%). In all the scenarios the FEC rate is about 10% of the media rate and we observe recoveries due to congestion losses in this scenario, with 10-20% of the protected lost packets are recovered. Some packets were lost in bursts, in these cases a parity FEC scheme cannot provide additional protection.

5.4 Multiple RTP flows competing with multiple TCP flows

In this simulation experiment, we add another RTP flow to the scenario in Section 5.3, i.e., the second flow competes for capacity with the other flows at the bottleneck link. All other parameters including the network topology, link characteristics, and traffic source properties remain the same (see Figure 5c for details).

The goodput for each RTP flow is comparable to the other for all three congestion control algorithms, i.e., each RTP flow is fair to the other and leaves the competing RTP flow enough bandwidth to provide a similar user experience. However, what differs between them is the interaction with the short TCP flows. At low bottleneck link delays, the TCP Fair Share (TFS) for TCP flows competing with FBRA and C-NADU is around or greater than 100% (see Table 3), which shows that both algorithms move out of the way of the TCP flows. However, RRTCC is more competitive and allows TCP flows only 55-80% of their fair share, which affects the completion times for these TCP flow sand may raise some doubts about its fairness. Our observations show that TCP flows competing with RRTCC will take 2-3 times longer than TCP flows competing with C-NADU or FBRA (compare the average TCP throughputs in Table 3).

Table 3: Overall metrics for 2 RTP Flows competing with 10 TCP flows

Delay		Metric	FBRA		RRTCC		C-NADU	
			avg.	σ	avg.	σ	avg.	σ
50ms	Flow 1	Goodput [kbps]	571.21	94.42	1640.29	37.625	742.63	100.55
		Loss rate [%]	2.1	0.21	5.65	0.09	1.2	0.12
		No. of lost frames	746.83	94.84	7787.67	124.83	801	68.11
	Flow 2	Goodput [kbps]	520.08	84.33	1750.09	65.45	748.65	72.95
		Loss rate [%]	2.4	0.18	5.48	0.30	1.17	0.14
		No. of lost frames	674.5	74.99	8140.33	420.82	787.67	128.85
	TCP	Throughput [kbps]	674.28	4.78	389.53	36.65	786.13	37.80
		Fair share [%]	134.86	0.96	77.9	7.78	152.7	2.92
100ms delay	Flow 1	Goodput [kbps]	745.42	87.85	1782.71	54.49	811.52	89.20
		Loss rate [%]	1.43	0.2	5.42	0.33	0.69	0.15
		No. of lost frames	566.53	77.02	8112.67	480.53	484	137.49
	Flow 2	Goodput [kbps]	691.8	83.58	1904.13	46.69	879.89	140.26
		Loss rate [%]	1.67	0.18	5.3	0.17	0.77	0.14
		No. of lost frames	521.57	68.05	8579.67	214.22	578.33	115.8
	TCP	Throughput [kbps]	466.21	3.26	281.7	23.51	547.63	32.77
		Fair share [%]	93.24	0.65	56.2	7.27	109.4	3.91
240ms delay	Flow 1	Goodput [kbps]	260.26	100.42	2145.09	18.96	1299.87	364.3
		Loss rate [%]	0.59	0.19	6.26	0.26	0.47	0.19
		No. of lost frames	155.47	52.85	11001.3	440.96	514.67	104.71
	Flow 2	Goodput [kbps]	287.41	140.73	2231.51	14.01	1039.08	277.35
		Loss rate [%]	0.71	0.19	6.02	0.06	0.54	0.09
		No. of lost frames	174.37	61.59	11234.67	120.29	459.67	119.68
	TCP	Throughput [kbps]	299.91	1.94	101.6	6.52	291.22	37.05
		Fair share [%]	59.98	0.39	20.32	0.10	58.24	0.41

FBRA's performance is comparable to C-NADU in the low delay scenario but due to its delay sensing behavior performs conservatively in the high delay scenario ($240ms$). The FEC rate is 10% of the media rate and the FEC recovers $FFRE =$ 13-17% of the protected media packets. The FEC is more accurate for congestion control in the low delay scenario ($FFRC =$ 88-91%) than for high delay ($240ms$) scenario ($FFRC =$ 81-82%).

5.5 Real World

Ns-2 simulations only take the network aspects into account, such as packet loss due to router drop, bit-error loss, queuing delay etc., ignoring multimedia aspects like, the Group of Picture (GOP) structure, different types of video frames (I- P-, or B-frames), etc. Since no real media packets are sent in *ns-2*, the simulations can neither

measure the PSNR nor determine the Mean-opinion Score (MoS) for evaluating the multimedia quality experienced by the user. Simulations in *ns-2* also ignore the performance issues with real-life systems, like OS kernel, device drivers, etc. Therefore, we also conduct experiments in a real-world setting.

Our video call application is built on top of open-source libraries: Gstreamer[6], x264[7] and JRTPLib[8]. We have also extended the JRTPLib to generate and decode FEC, perform congestion control, generate RTCP XRs, and be compliant with RFC4585 RTCP timing rules. In addition, the FEC module is fully compatible with RFC 5109 [41]. The application can encode and decode files or take input from a webcam and render on the screen. To evaluate performance, we use the *"News"* video sequence[9] in VGA frame size and 15 FPS. Details concerning the system design are presented in section 6.

Due to heterogeneity in the networks, interactive multimedia applications use a short GOP structure (≤ 5) [19, 23,38,42], helping overcoming variability in the available end-to-end capacity, bit-error losses, etc. By using FEC for congestion control, our experiments show that the congestion control algorithms can use a very long GOP structure. Since FBRA's maximum $FEC_{interval}$ is 14 packets, we propose using a GOP of a similar size. In addition, the encoding/decoding complexity of a stream using a larger GOP is much smaller than of a stream using a smaller GOP and for the same encoding rate P-frames can be less compressed. Hence, a larger GOP stream should lead to a better PSNR and improved energy consumption [43].

We use Dummynet[10] [44] to emulate the variation in link capacity, latency and/or losses in our testbed. Using Dummynet, we evaluate the FBRA algorithm in two scenarios: 1) two RTP flows compete on a bottleneck link, 2) RTP flow competes against short TCP flows on a bottleneck link. In both scenarios, the bottleneck link capacity is $1Mbps$ and the end-to-end path latencies are $50ms$ and $100ms$. The TCP traffic model is identical to the one implemented in the simulations. Finally, we initiate a video call between a Linux machine at Aalto University (Helsinki) and an Amazon EC2 virtual machine over the public Internet. For deriving statistical significance, each scenario is run 10 times.

To compare our results with RRTCC in the real-world, we use Google's Chrome browsers and the video source uses the same test YUV file instead of a webcam. To evaluate the performance, the browsers send the media streams through our *dummynet* testbed.

[6]http://gstreamer.freedesktop.org/
[7]http://www.videolan.org/developers/x264.html
[8]http://research.edm. uhasselt.be/˜jori/
[9]http://xiph.org
[10]http://info.iet.unipi.it/ luigi/dummynet/

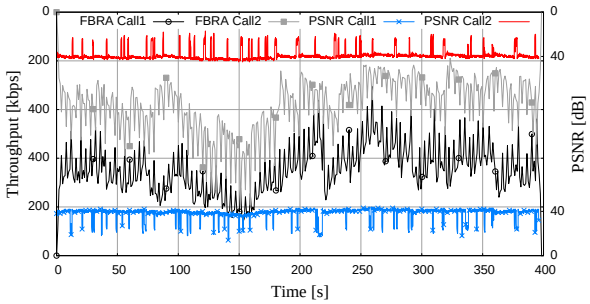
Table 4: Dummynet: Two RTP flows on a bottleneck link

	Metric	Call 1		Call 2	
		avg.	σ	avg.	σ
50ms delay	**Goodput [kb/s]**	375.39	88.25	348.77	83.64
	Loss rate [%]	1.21	0.19	1.39	0.69
	FEC rate [kb/s]	12.24	1.64	11.78	1.39
	No. of lost frames	72.5	7.8	80.1	27.85
	No. of FEC protected lost frames	15.3	3.44	14.6	2.73
	FFRE [%]	7.41	9.76	2.2	3.58
	FRCC [%]	83.19	2.55	83.39	2.62
	PSNR [dB]	38.08	2.1	37.7	1.53
100ms delay	**Goodput [kb/s]**	295.33	48.27	351.1	63.4
	Loss rate [%]	3.15	0.93	2.33	0.87
	FEC rate [kb/s]	10.7	0.68	11.69	1.52
	No. of lost frames	174.6	92.44	133.8	42.33
	No. of FEC protected lost frames	3.0	2.1	4.1	3.14
	FFRE [%]	0.0	0.0	1.54	4.62
	FRCC [%]	82.99	2.16	84.69	3.79
	PSNR [dB]	35.64	1.17	37.32	1.65

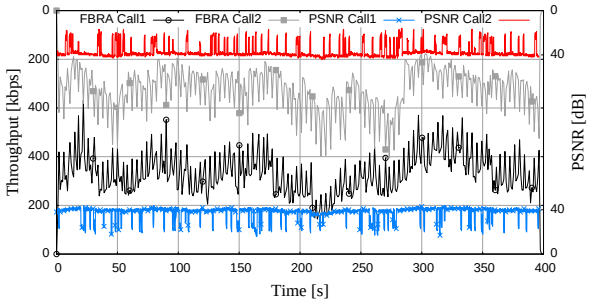
Dummynet two RTP flows competition scenario: The RTP flows using FBRA are fair to one another (see Table 4) and the goodput results are similar in magnitude when compared to the simulation results. The loss rate is much lower than in the simulation results. Hence, the frame recoveries (FFRE) are also lower. The difference in the goodput of the calls in the two delay cases is about 30-50 *kbps*, but the PSNR of these calls are similar (see Table 4). Therefore, we conclude that small rate variations ($\approx$ 10%-20%) have little bearing on the quality of the call. The FEC rate is about 10-12$kbps$ and is smaller than the rate achieved in the simulations, which means that actual overhead introduced by FEC addition is also smaller. This also implies that the rate control algorithm remains longer in the *STAY* state, thus avoiding abrupt changes to the encoding rate, which is detrimental for user experience [2]. On the other hand, RTT variations in the physical networks causes the accuracy of using FEC for rate control (FRCC) to be also lower than in the simulations, and is around 83-84%.

RRTCC calls have a throughput of 392 *kbps* ($\sigma = 120$) and 545 *kbps* ($\sigma = 130$) for 50 *ms* delay scenario and approximately 10-25 *kbps* lower for the 100 *ms* delay scenario. These results are comparable to the goodput achieved by FBRA. However, the higher standard deviation in the bandwidth measurement of the RRTCC calls denotes higher variation during the session and hence, poor user experience.

Dummynet RTP vs. TCP flows competition scenario: The RTP flow competes well against short TCP flows achieving an average goodput of 302 *kbps* and 280 *kbps* in the $50ms$ and $100ms$ delay scenario, respectively (see Table 5). The TCP flow achieves a throughput of around 600 *kbps* on average, the loss rate is around 4%,



(a) OWD=50ms



(b) OWD=100ms

Figure 9: Shows the goodput of two RTP calls sharing a common bottleneck. To illustrate amount of empty link capacity and how two flows push one another, we plot one of them on the reverse axis. The end-to-end path capacity is $1Mbps$ in both delay scenarios and delays are $50ms$ and $100ms$. The plot also shows the PSNR variation for the two calls (on the minor Y-axis).

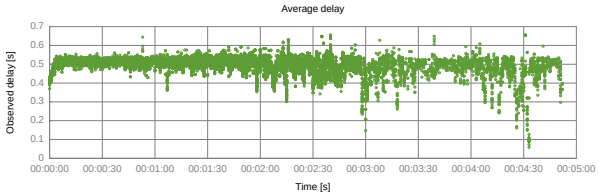
Table 5: Dummynet: An RTP flow sharing a bottleneck link with short TCP flows

Metric	**50ms delay**		**100ms delay**	
	avg.	σ	avg.	σ
Goodput [kb/s]	302.24	87.07	280.97	92.15
Loss rate [%]	4.24	0.89	4.1	0.58
FEC rate [kb/s]	13.6	2.15	12.58	2.18
No. of lost frames	154.6	16.56	170.9	12.38
No. of FEC protected lost frames	38.0	6.08	23.7	8.99
FFRE [%]	6.62	4.01	6.77	5.8
FRCC [%]	83.32	2.7	84.06	2.81
PSNR [dB]	35.62	1.49	34.7	2.26
TCP throughput [kbit/s]	612.22	48.45	575.11	45.67

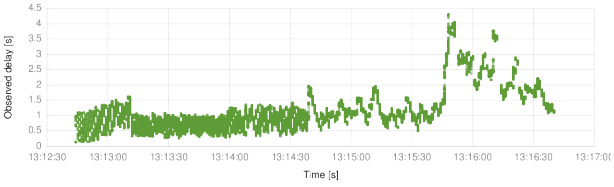
which is higher than in the simulation results. This effect mainly arises from the variations in RTT, which leads to a higher amount of packet discards at the receiver. Frame recoveries are also less frequent with *FFRE* $\approx$ 6-7%. PSNR results obtained in both scenarios are very similar ($\approx 35dB$), the PSNR is lower for the $100ms$ delay scenario because the average goodput is also a bit lower in this case. Similar to the previous scenario, the FEC rate is about 12-13$kbps$ and the *FRCC*$\approx$ 83-84%. This is lower than in the ns-2 simulations, and is again due to the RTT variations, which are not present in the simulation environment.

RRTCC calls have a throughput of 203 *kbps* ($\sigma = 26$) in the 50 *ms* delay scenario, with TCP throughput of 761 *kbps* ($\sigma = 238$). In the 100 *ms* case, RRTCC obtains 189 *kbps* ($\sigma = 23$), while TCP reaches 867 *kbps* ($\sigma = 236$).

Higher standard deviation is not the only metric in which FBRA outperforms RRTCC. While RRTCC flows achieve high throughput, they also induce higher packet delay. Figure 10 illustrates the variation in the observed packet delay: 1) when two RRTCC flows compete with one another, and 2) when an RRTCC flow competes with a short TCP flow on a bottleneck link. We observe that the packet delay constantly exceeds the recommended *400ms* end-to-end delay [38] and sometimes spikes to values as high as *3s* (see figure 10b). On the other hand, FBRA always maintains packet delay below *400ms*, and discards all packets that arrive later. As a result, despite very comparable results in terms of throughput, FBRA provides better user experience, as the throughput variations are smoother, and the packet delay variation is lower.



(a) Two RTP competition scenario



(b) RTP vs. TCP competition scenario

Figure 10: shows the variation in packet delay for RRTCC congestion control. We observe packet delay much higher than the *100ms* link latency. Furthermore, the observed end-to-end delay rarely goes below the recommended *400ms*, and sometimes the delay spikes to *3s*, which makes video conversation effectively impossible. This is a representative plot of 10 successive runs using the Chrome browser on our testbed.

Call over the public Internet: By initiating a video call between a host on an Amazon EC2 instance and a

machine at the university, we measure the performance of the FBRA in the public Internet. We observe varying results between each successive run, as the public Internet has varying amount of cross traffic. The goodput ranges between 100-700 *kbps* and the maximum loss rate does not exceed 1.5%. The PSNR of the calls also varies between 35-40 *dB*. Despite results diversity, we show that FBRA may work in the public Internet.

6 System considerations

Congestion control algorithms for multimedia communication are never used stand-alone, but are built into advanced systems where tight co-operation between multiple components is required. In this section, we present the design and implementation of the *Adaptive Multimedia System* (AMuSys).

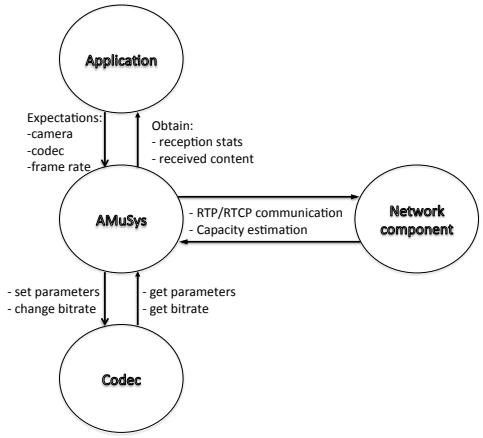


Figure 11: shows the design of the *Adaptive Multimedia System (AMuSys)*. The system provides APIs to communicate between the application, the network component and the codec.

6.1 System Description

AMuSys is made up of 3 sub-systems, namely, the application, codecs and the networking components. Figure 11 illustrates the integration of the application, codec, and the network subsystems in *AMuSys*. It also implements the control loops (presented in [45]) needed to design advanced congestion control algorithms that take into consideration the codec constraints along with the network-related parameters. Application developers, who may not be multimedia communication experts, may delegate the responsibility for the whole communication process to *AMuSys* by just specifying the desired application preferences. *AMuSys* defines three main interfaces, namely the application interface, the network interface, and the codec interface. Table 6 summarizes the methods offered by each interface.

The application interface allows an end-user application to specify its preferences by calling the `setPreferences()` method. The typical preferences are:

(1) codec type,
(2) expected frame rate,
(3) signal source device,
(4) expected display resolution.

These preference may also be a result of capability negotiation during session setup between the endpoints

The codec interface allows the *AMuSys* to modify the codec settings during an ongoing multimedia session. The interface is designed to be flexible and codec independent. Therefore, all codec parameters are set using the `setParam()` method, which takes 2 arguments, namely a key-value pair containing the name of the codec-parameter, and the updated value. Since the method is codec-independent, it is the responsibility of the systems integrator to use the correct parameter name and value. Using incorrect arguments causes a non-fatal error to be reported and the parameter is not modified. The interface also provides a `getParam()` method to be able to fetch the current value of a desired parameter. In addition, the interface explicitly provides a method to obtain and update the media encoding rate. The `setBitrate()` method updates the encoding rate, while the `getBitrate()` method can be used to check if the requested rate has been actually imposed by the encoder, or to monitor how quickly the codec is able to converge its encoding rate to the requested value.

The network interface provides functions to modify the network parameters of the multimedia session. The features are divided into two main categories. The first one comprises callback functions that are invoked in the *AMuSys* control unit by the network component (e.g., RTP library) when a specific event occurs. For instance, it may process the RTCP RR in the `onReceivedRTCPPacket()` callback. The most important callback functions are listed in the table 6. The second category consists of basic session settings that can be updated during runtime. These settings include:

(1) session profile choice (AVP [11] vs. AVPF [13]),
(2) RTCP interval modifications,
(3) RTCP extensions choice (e.g., XR blocks [15],
(4) rapid timestamp synchronization (RFC 6051 [46]).

Since the *AMuSys* is able to access information from the other interfaces, it has a better understanding of the

Name	Input	Output	Description
	Application interface		
`setPreferences()`	`dictionary`	-	Specifies end-user application requirements
	Codec interface		
`getParam()`	`name`	`value`	Gets value of parameter name
`setParam()`	`name, value`	`true/false`	Sets parameter name to value
`getBitrate()`	-	`value`	Gets current bitrate
`setBitrate()`	`value`	-	Sets bitrate to $value$
	Network interface		
`onReceivedRTPPacket()`	`payload, source address, timestamp`	-	Invoked on RTP packet reception
`onReceivedRTCPPacket()`	`payload, source address, timestamp`	-	Invoked on RTCP packet reception
`onSendRTPPacket()`	`payload, timestamp`	-	Invoked after RTP packet sending
`onSendFECPacket()`	`payload, timestamp`	-	Invoked after FEC packet sending
`onReceivedFECPacket()`	`payload, source address, timestamp`	-	Invoked on FEC packet reception
`enableFEC()`	`true/false`	-	Switches on/off FEC
`setFECScheme()`	`Scheme specific input`	-	Updates used FEC scheme
`setSessionParams()`	`dictionary`	-	Specifies session parameters
`sendEarlyRTCPReport()`	-	-	Sends early RTCP packet

Table 6: Methods exposed by the application, codec and network subsystems of *AMuSys*.

current system state, i.e., the current network, codec and application state, and thus is able to make better decisions for providing good quality user-experience.

7 Conclusion

In this paper, we show that FEC can be applied not only for error resilience, but can also be used for congestion control. We present a new congestion control algorithm (FBRA) that incorporates FEC packets for probing for available capacity. Performance of FBRA is compared with two other congestion control algorithms, namely, RRTCC and C-NADU. Our simulations show that RRTCC and C-NADU make opposite performance trade-offs (higher capacity instead of lower packet loss rate, and vice-versa). The FBRA algorithm evaluation shows that it can perform at a trade-off point between the other two algorithm. FBRA on average has better goodput than C-NADU and better packet loss rate than RRTCC. We also note that FBRA's performance drops at high e2e delay ($240ms$), because FBRA uses OWD to sense congestion. Also the applicability of FEC for congestion control reduces at high link delays, which is visible in worsening FRCC metric.

Furthermore, we evaluate the performance of FBRA and RRTCC in the real-world scenarios. We show that despite obtaining comparable throughput, FBRA provides users with enhanced user experience, as its packet delay variation and goodput variations are far lower.

Finally, we measure the performance of FBRA on the public Internet and show that the algorithm can be successfully applied. Since using FEC increases error resilience, we are able to increase the GOP size, which reduces the encoding and decoding complexity without affecting the user-experience.

As our concept is targeted to work on top of a congestion control unit, we believe that it can be applied not only to the FBRA algorithm, but also to any other rate control algorithm. This idea can be realized by adding an extra FEC subsystem to the congestion controller. For instance in RRTCC, when it receives a TMMBR message for increasing the rate, it can allocate the difference in the current rate and the new rate to FEC.

In the future work, we envision exploring incorporation of more complex FEC schemes. As the FEC frame recovery metric still shows room for improvement, we believe that application of different FEC scheme can provide useful gain to the overall performance. Furthermore, we are also interested in applying our concept in challenging heterogeneous environments where FEC features can prove to be particularly useful.

References

[1] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the Internet," *in Proc. of Communications of the ACM*, vol. 55, pp. 57–65, Jan 2012.

[2] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz, "Subjective Impression of Variations in Layer Encoded Videos," in *Proc. of IWQoS*, 2003.

[3] C. Jennings, T. Hardie, and M. Westerlund, "Real-time communications for the web," *IEEE Communications Magazine*, vol. 51, no. 4, pp. 20–26, April 2013.

[4] C. Perkins and V. Singh, "RTP Congestion Control: Circuit Breakers for Unicast Sessions," 2012, IETF Internet Draft.

[5] V. Singh, S. McQuistin, M. Ellis, and C. Perkins, "Circuit Breakers for Multimedia Congestion Control," in *Proc. of IEEE Packet Video 2013*, 2013.

[6] J. Devadoss, V. Singh, J. Ott, C. Liu, Y.-K. Wang, and I. Curcio, "Evaluation of Error Resilience Mechanisms for 3G Conversational Video," in *Proc. of IEEE ISM*, 2008, pp. 378–383.

[7] Y. Wang and Q.-F. Zhu, "Error control and concealment for video communication: a review," *in Proc. of the IEEE*, vol. 86, no. 5, pp. 974 –997, may 1998.

[8] Y. Wang, S. Wenger, J. Wen, and A. Katsaggelos, "Error resilient video coding techniques," *in Proc. of IEEE Signal Processing Magazine*, vol. 17, no. 4, pp. 61 –82, jul 2000.

[9] J. Evans, A. Begen, J. Greengrass, and C. Filsfils, "Toward lossless video transport," *in Proc. of IEEE Internet Computing*, vol. 15, no. 6, pp. 48 –57, nov.-dec. 2011.

[10] "The Network Simulator NS-2," http://www.isi.edu/nsnam/ns/.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550 (INTERNET STANDARD), Internet Engineering Task Force, Jul. 2003, updated by RFCs 5506, 5761, 6051, 6222, 7022. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[12] E. Brosh, S. A. Baset, D. Rubenstein, and H. Schulzrinne, "The Delay-Friendliness of TCP," in *Proc. of ACM SIGMETRICS*, 2008.

[13] 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), Internet Engineering Task Force, Jul. 2006, updated by RFC 5506. [Online]. Available: http://www.ietf.org/rfc/rfc4585.txt

[14] C. Perkins, "On the Use of RTP Control Protocol (RTCP) Feedback for Unicast Multimedia Congestion Control," 2013, IETF Internet Draft.

[15] T. Friedman, R. Caceres, and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)," RFC 3611 (Proposed Standard), Internet Engineering Task Force, Nov. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3611.txt

[16] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," *in Proc. SIGCOMM CCR*, vol. 30, pp. 43–56, 2000.

[17] L. Gharai and C. Perkins, "RTP with TCP Friendly Rate Control," March 2011, (Work in progress).

[18] A. Saurin, "Congestion Control for Video-conferencing Applications," Master's Thesis, University of Glasgow, December 2006.

[19] V. Singh, J. Ott, and I. Curcio, "Rate adaptation for conversational 3G video," in *Proc. of INFOCOM Workshop*, Rio de Janeiro, BR, 2009.

[20] R. Rejaie, M. Handley, and D. Estrin, "Rap: an End-To-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet," in *Proc. of INFOCOM*, Mar 1999.

[21] L. Gharai and C. Perkins, "Implementing Congestion Control in the Real World," in *Proc. of ICME '02*, vol. 1, 2002, pp. 397 – 400 vol.1.

[22] H. Vlad Balan, L. Eggert, S. Niccolini, and M. Brunner, "An Experimental Evaluation of Voice Quality Over the Datagram Congestion Control Protocol," in *Proc. of IEEE INFOCOM*, 2007.

[23] H. Garudadri, H. Chung, N. Srinivasamurthy, and P. Sagetong, "Rate Adaptation for Video Telephony in 3G Networks," in *Proc. of PV*, 2007.

[24] V. Singh, J. Ott, and I. Curcio, "Rate-control for Conversational Video Communication in Heterogeneous Networks," in *in Proc. of IEEE WoWMoM Workshop*, SFO, CA, USA, 2012.

[25] X. Zhu and R. Pan, " NADA: A Unified Congestion Control Scheme for Real-Time Media," http://tools.ietf.org/html/draft-zhu-rmcat-nada-01, 2013, IETF Internet Draft.

[26] P. O'Hanlon and K. Carlberg, "Congestion control algorithm for lower latency and lower loss media transport," http://tools.ietf.org/html/draft-ohanlon-rmcat-dflow, 2013, IETF Internet Draft.

[27] Ł. Budzisz, R. Stanojević, A. Schlote, F. Baker, and R. Shorten, "On the fair coexistence of loss-and delay-based tcp," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 6, pp. 1811–1824, 2011.

[28] H. Alvestrand, S. Holmer, and H. Lundin, "A Google Congestion Control Algorithm for Real-Time Communication on the World Wide Web," 2012, IETF Internet Draft.

[29] S. Wenger, U. Chandra, M. Westerlund, and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)," RFC 5104 (Proposed Standard), Internet Engineering Task Force, Feb. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5104.txt

[30] L. D. Cicco, G. Carlucci, and S. Mascolo, "Experimental Investigation of the Google Congestion Control for Real-Time Flows," in *Proc. of ACM SIGCOMM 2013 Workshop on Future Human-Centric Multimedia Networking*, 2013.

[31] V. Singh, A. A. Lozano, and J. Ott, "Performance Analysis of Receive-Side Real-Time Congestion Control for WebRTC," in *Proc. of IEEE Packet Video 2013*, 2013.

[32] W. Zhu and Q. Zhang, "Network-Adaptive Rate Control With Unequal Loss Protection For Scalable Video Over Internet," *in Proc. of Circuits and Systems*, 2001.

[33] Q. Zhang, W. Zhu, and Y.-Q. Zhang, "Network-adaptive rate control and unequal loss protection with tcp-friendly protocol for scalable video over internet," *in Proc. of The Journal of VLSI Signal Processing*, vol. 34, pp. 67–81, 2003.

[34] ——, "Network-adaptive rate control with tcp-friendly protocol for multiple video objects," in *Proc. of ICME*, 2000.

[35] J. Ott, I. Curcio, and V. Singh, "RTP Control Protocol (RTCP) Extended Reports (XR) for Run Length Encoding (RLE) of Discarded Packets," 2011, IETF Internet Draft.

[36] B. Ngamwongwattana and R. Thompson, "Sync & Sense: VoIP Measurement Methodology for Assessing One-Way Delay Without Clock Synchronization," *in Proc. of IEEE Transactions of Instrumentation and Measurement*, vol. 59, no. 5, pp. 1318–1326, 2010.

[37] R. Brandenburg, K. Gross, Q. Wu, F. Boronat, and M. Montagud, "RTCP XR Report Block for One Way Delay metric Reporting," 2012, IETF Internet Draft.

[38] 3GPP S4-080771, "MTSI Video Dynamic Rate Adaptation: Evaluation Framework," Oct. 2008.

[39] R. Jesup, "Congestion control requirements for rmcat," 2013, IETF Internet Draft.

[40] V. Singh and J. Ott, "Evaluating congestion control for interactive real-time media," 2013, IETF Internet Draft. [Online]. Available: http://tools.ietf.org/html/draft-singh-rmcat-cc-eval

[41] A. Li, "RTP Payload Format for Generic Forward Error Correction," RFC 5109 (Proposed Standard), Internet Engineering Task Force, Dec. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5109.txt

[42] I. Curcio and D. Leon, "Application rate adaptation for mobile streaming," *Proc. of IEEE WOWMOM*, 2005.

[43] M. Wien, H. Schwarz, and T. Oelbaum, "Performance analysis of svc," *in Proc. of IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1194 –1203, sept. 2007.

[44] M. Carbone and L. Rizzo, "Dummynet revisited," *in Proc. of ACM SIGCOMM CCR*, Jan 2010.

[45] V. Singh, J. Ott, and C. Perkins, "Congestion Control for Interactive Media: Control Loops & APIs," in *IAB/IRTF Workshop on Congestion Control for Interactive Real-Time Communication*, July 2012. [Online]. Available: http://csperkins.org/publications/2012/07/iab-cc-workshop.pdf

[46] C. Perkins and T. Schierl, "Rapid Synchronisation of RTP Flows," RFC 6051 (Proposed Standard), Internet Engineering Task Force, Nov. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc6051.txt