FRACTaL: FEC-based Rate Control for RTP

Balázs Kreith
callstats.io · University of Debrecen
balazs@callstats.io

Varun Singh
callstats.io
varun@callstats.io

Jörg Ott
TU München · callstats.io
ott@in.tum.de

ABSTRACT

We propose a new rate control algorithm for interactive real-time multimedia traffic, the FRACTaL algorithm. In our approach, the endpoint sends Forward Error Correction (FEC) packets not only for better error-resilience but also to probe for available bandwidth. The sender varies the amount of FEC to meet the sending rate calculated by the congestion control without changing the media rate. We evaluate our proposal in an emulated networking environment, using a set of reference test scenarios and compare it to the SCReAM congestion control algorithm. We find that FRACTaL performs better than SCReAM when competing with TCP flows, i.e., it is able to obtain its "fair" share. In other (non-TCP) scenarios, it achieves lower loss rates at comparable path utilization and queuing delay, i.e., FRACTaL delivers better and consistent media quality.

1 INTRODUCTION

Web-based real-time communication (WebRTC) [16, 43] is stimulating the growth of conversational multimedia communication on the Internet, in addition to the already dominant multimedia streaming. This increase in real-time multimedia traffic has resulted in a renewed interest in congestion control for conversational multimedia systems. Real-time communications imposes limits on the end-to-end packet delivery latency to stay within the recommended mouth-to-ear delay of 150 ms to maintain a fluent real-time conversation between participants [17].

The best-effort nature of the Internet yields communication paths with time-varying characteristics such as varying latency, available capacity of bottleneck links due to cross-traffic, or variable link data rates in wireless networks, among others. This usually manifest in either packets being delayed or lost, which results in poor media quality. To cope with losses, multimedia transport protocols may use a number of different—media/codec-specific or generic—mechanisms to repair packet losses or limit their impact [7, 27, 44]. Conversational real-time media can often not afford to use retransmissions because loss detection and retransmission may make a media packet miss its playout point at the receiver. Even though receivers apply techniques to conceal errors, missing packets may result in choppy audio and frozen, blurred, or otherwise improperly rendered video frames [13].

Alternatively, Forward Error Correction (FEC) improves the reliability by adding redundant packets (carrying FEC symbols sent as a secondary flow) to the media stream packets (source symbols of the original, primary flow). If some original packets are lost, symbols from the primary and secondary stream can be combined to recover those. This comes at a cost: besides a modest added latency in case of recovered packets, adding FEC yields an increased data and packet rate. While a rate increase is generally not desirable as it may increase congestion and thus cause further losses, it is acceptable if the codec output is anyway lower than the target data rate determined by a congestion control algorithm.

Transport protocols seek to prevent self-inflicted packet losses due to a media sending rate exceeding the end-to-end path capacity. The congestion control algorithms aim at determining the available capacity and maximize its utilization. While reducing the transmission rate in case of detecting congestion is a straightforward reaction, increasing the sending rate to find the current capacity is more difficult and comes at the risk of building up queues and creating losses if the sender over-estimates the capacity. While a congestion control algorithm may output a (increased) target sending rate, a media source may face additional sending rate limitations for two reasons: 1) A codec may not be able to switch the encoding of the media stream (immediately) to arbitrary bit rates and thus limit the precision of rate adaptation algorithms. 2) Adapting the media encoding rate too often or in large steps degrades the user-perceived media quality [47]. Therefore, gradual changes in the encoding rate are generally advised.

In this paper, we present a novel way to exploit the properties of FEC, using extra capacity for redundancy to create robustness against packet losses, to cope with the aforementioned limitations of media codecs when adapting the encoding rate. The result is a FEC-based congestion control scheme, FRACTaL, that extends our concept of using FEC for real-time media congestion control introduced in [20, 38] in several ways. The main idea behind using FEC for congestion control is to enable FEC alongside the media stream and use the redundancy to probe the path for available capacity. If the path conditions remain stable after the FEC stream is added, in the subsequent bandwidth estimation, the media encoding rate is increased by the amount of data (media and FEC) added in the previous bandwidth estimation. The main advantage of using FEC is that the applied congestion control algorithm can be more aggressive in probing for the available path capacity, as the improved reliability compensates for possible errors resulting from link overuse.

We make the following contributions: We introduce the Flexible and Adaptive FEC for RTP Congestion Control (FRACTaL), designed to meet the requirements of the RTP Media Congestion Avoidance Techniques (RMCAT) Working Group (WG) [15]. FRACTaL overcomes the two issues encountered by FEC-based Rate Adaption (FBRA) [20]: FBRA is conservative when ramping up the sending rate and is not able to compete (well) against TCP flows. We implement the proposed FRACTaL algorithm in Gstreamer and evaluate it based upon the RMCAT scenarios [31]. We finally compare

MM '17, October 23–27, 2017, Mountain View, CA, USA

ACM ISBN 978-1-4503-4906-2/17/10…$15.00
https://doi.org/10.1145/3123266.3123373

the performance of our FEC-based congestion control algorithm for WebRTC to the Self-Clocked Rate Adaptation for Multimedia (SCReAM) [19], a congestion control algorithm implemented and deployed in OpenWebRTC [23].

2 BACKGROUND AND RELATED WORK

The Real-time Transport Protocol (RTP) [32] is used to deliver media in real-time applications. In addition to media data, RTP may carry repair and redundant packets (such as retransmissions, FEC, comfort noise, etc). RTP is complemented by the Real-time Transport Control Protocol (RTCP), which conveys statistics measured at the sending and receiving endpoints. Sender Reports (SR) inform the receiver about the packets and bytes sent and provide timestamps for RTT measurements and synchronization across media streams. Receiver Reports (RR) provide feedback about received and lost packets, jitter, and RTT.Th e information in the basic reports are coarse-grained but eXtention Reports (XR) allow conveying more detailed reception and loss statistics [9].

RTP is a general purpose real-time transport that can support arbitrary multimedia applications, but its main use is for interactive multimedia communication such as video conferencing. Media streaming commonly uses TCP (usually with adaptive HTTP streaming on top). But the aforementioned strict latency requirements for conversational multimedia limit TCP's usefulness; studies have shown that media experience over TCP deteriorates when the one-way delay exceeds 50 ms. Instead, RTP over UDP is used to maintain interactivity [1]. While TCP is not preferred, in networks where UDP packets are dropped orfi ltered by middleboxes, TCP may be used as a fallback: it is more important to have media connectivity at all and to trade-off smooth media experience [18].

Implementing congestion control for transport protocols requires frequent feedback from the receiver to the sender: per-packet, per-frame, or per-RTT [26]. Baseline RTP [32], designed with multicast-based group communication in mind, limits the RTCP reporting interval to a minimum of 5 ± 2.5 s and to 5% of the media bit rate. To enable repair and ultimately also congestion control, a new RTP profile (AVPF) removes the lower time bound [25], enabling RTP-based circuit breakers [28, 37] and congestion control.

Numerous congestion control algorithms have been proposed over the years, the most prominent is TCP-Friendly Rate Control (TFRC) [8]. It can be adapted to RTP without any changes to RTP, except that the TFRC equation expects loss events and the RTCP reports fractional loss.Th e loss events are typically fewer compared to loss packets, because burst losses are only counted as one loss event [10]. Nonetheless, TFRC produces afl uctuating media rate, observed as a short-term sawtooth. Changing media characteristics by too much in a very short period of time results in a bad quality of experience [47]. RAP [29], a delay based congestion control uses additive increase multiplicative decrease (AIMD), which also produces a saw-tooth. Inspired by the approach in [4], DFlow [22] uses two control algorithms, one for competing against media traffic (delay-based) and another for competing against TCP cross-traffic (loss-based). Basically, it switches between the delay- and loss-based congestion controls when the observed RTT and loss is above a threshold. NADA [6, 46] also uses delay- and loss-based mode, except that NADA relies on queuing delay by estimating the per-packet inter-arrival time. NADA uses 100 ms as the queuing delay threshold, i.e., irrespective of the network delay, if the packet has an estimated additional delay below 100 ms the algorithm uses the delay-based mode, otherwise the algorithm switches to the loss-based mode.

Several algorithms, Self-Clocked Rate Adaptation for Multimedia (SCReAM) [19] and Google Congestion Control (GCC) [12, 36] assume that the packet delay variation for an uncongested path can be modeled as a White Gaussian Noise (WGN), i.e., the median over a set of measurements can be used to estimate the actual network delay between peers. GCC determines the average queuing delay value for a group of packets by applying a Kalman Filter over all the jitter measurements [5]. In addition to the queuing delay estimate, GCC uses TFRC to estimate the available bandwidth, and does not reduce the sending rate until the estimate fractional loss is above 2%. SCReAM is a window-based and byte-oriented congestion control algorithm: it controls the sending rate based on the size of the congestion window (cwnd) and the number of bytes inflight. The congestion window is updated when an acknowledgment is received; cwnd is only decreased once per-RTT, on observing the first loss event, to not over-react and smoothen the loss over that measurement period. Additionally, SCReAM calculates the autocorrelation over the last 20–50 jitter measurements and compares it a queuing delay threshold to estimate if there is congestion.

FEC-based Rate Adaptation (FBRA) [20] was thefi rst algorithm to use redundant FEC packets to probe for additional capacity instead of increasing the media rate when a higher sending rate is estimated. FBRA uses a delay estimate to perform congestion control: the delay estimates are percentiles calculated over the entire lifetime of the session, i.e., 40% indicating a lower threshold, and 80% indicating the higher threshold.Th e intuition is that the sending rate ramps up more quickly when the estimated one-way delay is at the lower threshold and sending rate ramps up slowly when the one-way delay is closer to the higher threshold.

FRACTaL builds on FBRA, also using FEC for probing for additional capacity. But FRACTaL adapts its use of FEC to the observed packet loss patterns, uses an improved state machine to make FRACTaL ramp up quicker, and uses different windows for interpreting congestion cues.Th is makes it compete well with TCP traffic.

3 USING FEC FOR CONGESTION CONTROL

The normal use FEC is protecting a set of packets against losses. It adds redundant packets to allow recovering lost parts without the need for a retransmission. FEC is useful when extra capacity is available and the latency is high [7, 13].Th e fraction of added redundancy can be tailored to the intended protection level, e.g., 10% to cope with one packet lost in ten.

The main idea of using FEC for congestion control is simple: congestion control algorithms need to decide when to increase the sending rate but, by doing so, they risk that the extra packets transmitted exceed the available capacity, which may result in lost media packets. Increasing the rate using FEC instead of media packets allows the sender to probe for extra capacity without the risk of losses: if a packet (media or FEC) is lost, it can be recovered. If all the packets are received, the receiver will report an increase in reception rate and the sender can replace the FEC packets with extra media packets, thus increasing the sending rate safely.The

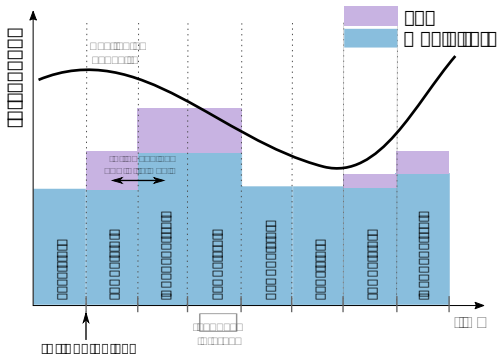


Figure 1: Using FEC for congestion control: If no congestion is observed, FEC is added to probe for additional bandwidth. If the combined sending rate does not cause congestion, the media rate is increased; otherwise it is reduced. Adapted from [20].

overall concept of using FEC for congestion control is shown in Figure 1.

Framework: Figure 2 shows the interaction between the congestion control algorithm, the RTP subsystems, and the Forward Error Correction (FEC) module [42]. At the sending endpoint, the congestion control module estimates the end-to-end path capacity based on the congestion cues carried in the RTCP feedback message [24, 26, 32]. If the estimated rate is higher than the previous estimation, the controller enables the FEC module, which creates redundant packets for protecting media stream and essentially increase the sending rate, thus probing the link capacity. At the receiving endpoint, the FEC module reconstructs the lost packets in the media stream from the cumulative packets received in the media and the FEC stream. If the packets are repaired the receiver generates the post-repair loss report (PRLR) for the corresponding RTP packets [14]. Consequently, the sender can use the information in the PRLR RTCP Report to calculate the effectiveness of FEC.

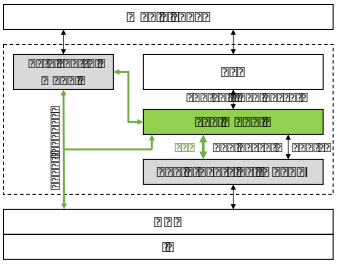


Figure 2: Interaction of Congestion Control and FEC Module.

State Machine: The efficiency of the congestion control algorithm using FEC depends on the decision when to and how much FEC to add. The high-level operation of FRACTaL is shown in Figure 3; its use of FEC is improved over FBRA [20]. FBRA also uses the four states STAY, PROBE, UP, and DOWN, which were mapped to KEEP, PROBE, INCREASE, and REDUCE in FRACTaL, respectively. We apply two major changes to the state transitions: **(1)** We remove

Figure 3: State machine of FRACTaL. State transitions occur upon the indicated conditions: if the reported metrics do not indicate congestion, a *stable conditions* exist; otherwise *unstable conditions* are recorded.

the transition from PROBE to KEEP in order to eliminate possible oscillations between the two states; in FRACTaL, the state needs to transition to REDUCE before going to KEEP from PROBE. **(2)** We added a transition between PROBE and INCREASE in order to ramp up the sending rate more quickly by skipping the KEEP state. Furthermore, we do not disable FEC at the INCREASE state in order to extend the protection against packet loss when the media bit rate is increased. Thus, we are able to perform congestion control and error resilience simultaneously, which is not supported in FBRA. By extending the period of FEC FRACTaL also gains a significantly better fair share compared to FBRA when competing with TCP.

3.1 Congestion cues

FRACTaL is a sender-driven congestion control algorithm[33]: the receiver collects, measures, and sends the statistics to the sender. The sender then uses these statistics to compute the new sending rate. The FRACTaL algorithm relies on several congestion cues: interval fractional loss [32], one-way delay [21], and bytes in flight [39, 41].

Receiver-side metrics: The receiver reports the metrics at regular intervals, with the RTCP interval depending on the current data rate. At video bit rates, it is possible to provide feedback per frame, i.e., every 33–150 ms [26]. We measure the *one-way delay* without clock synchronization by using the technique defined in [21] and report it in RTCP XR messages [3]. The *fractional loss* per interval is reported in the regular RTCP Receiver Report (RR) [32]; for more precision, we use the Extended Reports (XR) for discarded and lost packets [24, 40]. If the packets are repaired by an FEC packet, the receiver sends the post-repair loss report (PRLR) that contains the run-length encoding of all the repaired packets [14]. The Packet in Flight (PiF) is calculated by subtracting the Highest Sequence Number (HSN) reported in the RTCP RR and the last packet sent by the sender. With this and local packet size history, the sender can calculate the bytes in flight for these packets as well [39, 41].

Sender-side operation: The sender receives RTCP reports from the receiver at regular intervals [25]. It adds the congestion cues to sliding windows (of two different periods, see below), which it uses to filter and remove outliers. Received values are added if 1) they aren't outliers or 2) the sliding window contains too few (e.g., ¡10) data points. The rationale is that outliers should be filtered out unless there is a change in path conditions (towards congestion) so that what were outliers before turn into regular observations. The sender calculates relevant aggregated statistical measures over these slides windows and compares the aggregates to the most

recent values reported in the latest report to determine if the path is congested.

Sliding window and aggregate metrics:Th e sender uses a five second window for bytes infl ight because bytes infl ight can vary quite rapidly depending on the group of pictures (GoP) in the video (structure of I-, P-, and B-frames). In contrast, the window size for the one-way-delay (OWD) and fractional loss metric is 30 seconds tofi lter out transient network conditions.Th e sender calculates the following statistical measures:

(1) The median ($\widetilde{OWD}_t$) and standard deviation ($OWD_t^{(\delta)}$) of the one-way delay. Assuming that the packet delay variation for an uncongested path can be modelled as a white Gaussian process [12], the median and a standard deviation over a set of one-way delay measurements can be used to define a range that a consecutive OWD metric should operate in if the path is not congested.

(2) The 10%-ile ($FL_t^{(10th)}$) and standard deviation ($FL_t^{(\delta)}$) of the fractional loss. Assuming that an uncongested path may exhibit a non-zero loss rate we do not want to consider a path congested if the reported fractional loss appears in more than 90% in the reported metrics. We use an extra boundary in addition to 10%-ile derived from the standard deviation of the reported fractional loss.

(3) The 80%-ile of the bytes infl ight ($BiF_t^{(80th)}$). Assuming that the bytes in router queues increases over time if the path is not congested, we choose the 80%-ile of the measured value.This trims peaks possibly caused by sudden traffic spikes (I-frames) and provides reasonable information about the buffer capacity if the path become congested.Th is metric is used for pacing and for reducing the bitrate in case of congestion.

Congestion resholds:Th e sender calculates thresholds from the sliding window to determine if the path is congested.Th e relevant metrics are: the 10%-ile fractional loss metric and the median of the one-way delay value along with the standard deviation of these metrics.Th e thresholds for $OWD_t^{(\delta)}$ and $FL_t^{(\delta)}$ are calculated as follows:

$$OWD_t^{(thr)} = \widetilde{OWD}_t + max(OWD_t^{(\delta)} \times 2, OWD_{min}^{(thr)}) \quad (1)$$

$$FL_t^{(thr)} = min(FL_t^{(10th)} + FL_t^{(\delta)} \times 2, \quad FL_{max}^{(thr)}) \quad (2)$$

$OWD_{min}^{(thr)}$ is the minimum and $FL_{max}^{(thr)}$ is the maximum threshold for each metric.The $OWD_{min}^{(thr)}$ is set as a lower-bound for the one-way delay (e.g.: 30 ms).Th e algorithm does not consider the path to be congested if the measured value is below this threshold. This helps at the beginning of a real-time session, when the packet delay may vary substantially (e.g., a large I-frame may skew the metric). $FL_{max}^{(thr)}$ serves as upper-bound for the fractional loss metric (e.g.: 5%) that the algorithm must not exceed for proper media delivery.

4 THE FRACTAL ALGORITHM

FRACTaL is executed every time the sender receives an RTCP report with the feedback for the congestion cues defined above. We define one algorithm per state, each of which adjusts the transmission behavior and may cause a state transition. If a transition is invoked, it will occur upon receipt of the next RTCP report, i.e., the algorithm imposes a dynamic *validation delay*, T_{valid} (defined below), before changing states. If no RTCP report is received for $max(500ms, 3 \times RTT$, FRACTaL assumes congestion and enters the KEEP state.

FRACTaL maintains several further global time variables: t_{now} refers to the current time. t_{cong} indicates when congestion was last detected and $t_{settled}$ when an uncongested state is entered. t_{FEC} records when FEC transmission was enabled. RTT is the round-trip time as measured by RTCP.

In the following, we describe the algorithms for all four states.

KEEP: In the KEEP state the algorithm checks if the estimated sending rate R_{snd} causes congestion as measured per the congestion cues above. If two consecutive reports show hints for congestion, the algorithm executes an *undershoot* procedure (see below) and goes to the REDUCE state. If there is no congestion detected for a period of T_{valid} the algorithm calculates the current FEC interval, FEC_{int} , records the current time in t_{FEC}, performs a transition to the PROBE state, and starts sending FEC packets. If congestion is detected during an uncongested period t_{cong} is set; otherwise, if there is no sign of congestion, $t_{settled}$ is updated.

If the algorithm is executed again within twice RTT as measured by RTCP (i.e., $t_{now} - 2 \times RTT < t_{settled}$) the sender remains in the KEEP state and updates the sending rate smoothly, increasing or decreasing it according to $owd\_log\_corr = log_{10}(OWD)/log_{10}(\widetilde{OWD}_t)$.

PROBE: In the PROBE state, the algorithm checks if the added FEC rate does not cause congestion. If no congestion is detected for a certain period of time, the algorithm increases R_{snd} and transitions to the INCREASE state. Otherwise, it executes the *undershoot* procedure and moves to the REDUCE state.

INCREASE: In this state, the algorithm checks if the newly increased R_{snd} (cf. PROBE state) together with the FEC rate would not cause congestion. If this combined rate is approved, the FEC rate is reset, and the algorithm returns back to the PROBE state to continue probing with FEC on top of the new sending rate, R_{snd}. Otherwise, if congestion is detected, the algorithm performs *undershoot* and goes immediately to the REDUCE state.

REDUCE: In the REDUCE state, the algorithm determines a new capacity estimate. For calculating it, we assume that, without congestion, the receiver receives the mediafl ow at the transmitted rate.Th e sender will gather reports and estimate the capacity using a weighted average of the reported rate and the current sending rate.

If congestion is encountered, we assume that the amount of data queued in the network grows. Hence, we want to reduce the

Algorithm 1 KEEP

```
if OWD_t^(thr) < OWD_t OR FL_t^(thr) < FL_t then
    if congested = TRUE then
        Undershoot
        State ← REDUCE
    else
        t_cong ← t_now
        congested ← TRUE
    end if
else if congested = TRUE then
    State ← REDUCE
    congested ← FALSE
    t_settled ← t_now
else if t_now − 2 × RTT < t_settled then
    corr ← MAX(MIN(1.01, owd_log_corr), 0.99)
    R_snd ← R_snd × corr
else
    Start FEC
    State ← PROBE
    t_FEC ← t_now
end if
```

Algorithm 2 PROBE

```
if OWD_t^(thr) < OWD_t OR FL_t^(thr) < FL_t then
    Stop FEC, Undershoot
    t_cong ← t_now
    State ← REDUCE
else if t_FEC < t_now − T_valid^min then
    State ← INCREASE
    R_snd ← R_snd + FEC_rate
    t_incr ← t_now
end if
```

Algorithm 3 INCREASE

```
if OWD_t^(thr) < OWD_t OR FL_t^(thr) < FL_t then
    Stop FEC, Undershoot
    State ← REDUCE
    t_cong ← t_now
else if t_incr < t_now − T_valid then
    Restart FEC
    State ← PROBE
    FEC_started ← t_now
end if
```

Algorithm 4 REDUCE

```
if received_bitrate < R_snd × 0.5 then
    Undershoot
    t_cong ← t_now
else if OWD_t^(thr) < OWD_t OR FL_t^(thr) < FL_t then
    if t_cong < t_now − 1s then
        C_est ← receiver_rate
    else
        off ← (t_now − t_cong)/1.0s
        C_est ← receiver_rate × off + R_snd × (1 − off)
    end if
    R_snd ← (C_est − EQD/1.0s) × 0.9
else
    t_settled ← t_now
    State ← KEEP
end if
```

sending rate to drain this extra amount of queued data over time. We estimate this extra amount of queued data as $EQD = max(BiF_t - BiF_t^{80}, 0)$, i.e., the difference between the instantaneous bytes in flight and their long-term 80%-ile. In case of mild congestion, R_{snd} is set to be lower than the estimated capacity by subtracting EQD over a one second period. If the receiver rate is substantially lower than the R_{snd} (i.e., $< 0.5\times R_{snd}$), the algorithm assumes heavy congestion and executes the *undershoot* procedure.

If no congestion is observed, the algorithms transitions to the KEEP state. $t_{settled}$ used in the KEEP state is set to t_{now}.

Undershoot: The *undershoot* procedure [20] is used if congestion is detected to reduce the sending rate below the current capacity estimate C_{est}. To achieve this, the sender computes the new sending rate by subtracting *twice* the difference between the current sending rate and the current goodput (represented as EQD per second) and taking 90% of the obtained value, i.e., $R_{snd} = (C_{est} - 2 \times EQD/1.0s) \times 0.9$.

Scaling factors: FRACTaL is designed to be conservative when operating close to the capacity estimate and more aggressive when it senses that there is available network capacity.Thus, C_{est}, tracked by the algorithm, indicates the current (estimate of the) maximum available capacity along the path. To decide whether we are close to the capacity or not, the following scale factor, called offset from C_{est}, C_{est}^{off}, is calculated as $C_{est}^{off} = min(scale, 1)$ with $scale = \left(\frac{R_{snd}-C_{est}}{C_{est}\times\epsilon}\right)^2$ and $\epsilon \in (0, 1)$. $scale$ decreases exponentially as R_{snd} approaches $C_{est} \times \epsilon$. Based on the C_{est}^{off} value, a weighted average is used to determine $T_{valid} = (1 - C_{est}^{off}) \times T_{valid}^{min} + C_{est}^{off} \times T_{valid}^{max}$.

$T_{valid}^{min} = max(RTT, 100ms)$ and $T_{valid}^{max} = max(5 \times RTT, 1.5s)$ indicate the minimum and the maximum allowed values for the T_{valid}, respectively.Th e FEC interval, FEC_{int} , is calculated in a similar fashion:

$$FEC_{int} = (1 - C_{est}^{off}) \times FEC_{int}^{min} + C_{est}^{off} \times FEC_{int}^{max} \quad (3)$$

$FEC_{int}^{(min)}$ and $FEC_{int}^{(max)}$ indicate the minimum and the maximum values permitted for FEC_{int}, respectivey. FEC_{int} and T_{valid} depend on the value of C_{est}^{off}, which changes with C_{est} as calculated by the algorithm. It makes the FEC interval FEC_{int} larger and validation delay T_{valid} longer when operating near C_{est}.

Pacing: FRACTaL enforces pacing at the sender side. For pacing we want to give a reasonable value for the congestion window (*cwnd*) so the sending rate is not throttled. For that purpose we use the the 80%-ile of bytes infl ight and calculate *cwnd* depending on that value: $cwnd = BiF_t^{(80th)} \times 1.2$. $BiF_t^{(80th)}$ grows the congestion window when increasing the media rate, and shrinks it, if it senses congestion. When performing *undershoot*, *cwnd* shrinks faster to drain packets from the network, while the *cwnd* inflates quickly when the congestion disappears.

Flexible FEC: Parity codes are a form of systematic codes, wherein a repair packet is generated from a sequence of source packets (in this case, media packets). Parity codes generally use the eXclusive OR (XOR) operation to generate these repair packets [35].Th ere are two types of parity FEC codes: non-interleaved and interleaved. Non-interleaved repair packets are generated from consecutive source packets, while interleaved ones are generated from arbitrary set of source packets in the FEC interval.Th e FRACTaL algorithm uses both interleaved and non-interleaved parity FEC packets opportunistically based on the number of consecutive packet loss. **(1)** The sender selects a set of media packets to be protected within the FEC interval and determines the requested number of repair packets need to be generated; **(2)** Depending on the number of selected media packets and the requested number of repair packets, the FEC module (Figure 2) at the sender-side generates interleave and non-interleave FEC packets by applying XOR operation on the set of media packets; **(3)** The sender transmits the FEC packet(s) along with the media packets to the receiver.

5 PERFORMANCE EVALUATION

FRACTaL is implemented in the Gstreamer open source library, and can be used with applications built using OpenWebRTC [23]. We evaluate the performance of the algorithm based on the standardized testing scenarios proposed for testing RMCAT algorithms [31], based upon which we choose our evaluation parameters.

We measure congestion control performance in a test environment (Figure 4), in which the sender and the receiver run on several Linux machines connected via ethernet interfaces. However, Netem [11] applies constraints to the network interfaces to emulate the various characteristics described in each test scenario.The bottleneck is limited using tbf [11] and the queue length is always set to 300ms.Th e foreman video sequence [45] is used as the video

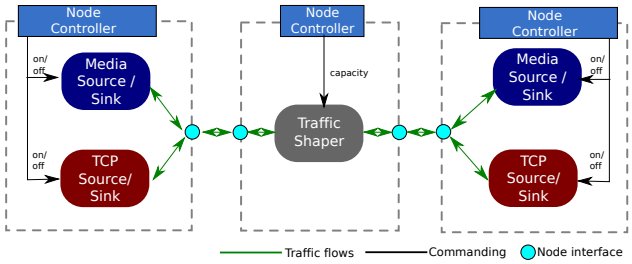


Figure 4: Performance Evaluation Setup.

		FCC	SCReAM
50ms	GP [kbps]	718.32 ± 16.28	844.32 ± 75.01
	LR [%]	0.39 ± 0.11	0.94 ± 0.41
	OWD [ms]	60.03 ± 1.78	72.76 ± 6.01
300ms	GP [kbps]	717.96 ± 38.39	803.05 ± 40.71
	LR [%]	1.09 ± 0.36	1.56 ± 0.33
	OWD [ms]	319.8 ± 8.29	325.61 ± 5.17

Table 1: Summary of RMCAT1 measurements. The FEC rate used by FRACTaL for 50 ms, and 300 ms is 110.54 ± 3.09 kbps, 88.73 ± 9.08 kbps

source with the VP8 codec [2]. We use *iperf(1)* (v2.0.5) to emulate the TCP cross traffic. For each scenario, we calculate the following metrics: goodput (GP), Loss Rate (LR), FEC Frame Recovery Efficiency *FFRE* [20]. When TCP is present, we calculate the TCP Fair Share (TFS) as well. Apart from these the forward path average queuing delays (OWD) is also measured and median is calculated over a one second long sliding window with a sample resolution of 100ms. Each scenario has a forward path one-way delay of 50 ms and 300 ms and each test have run 10 times. FRACTaL is compared to SCReAM, which is also implemented in Gstreamer [30].

RMCAT1—Single RTP on a Variable Capacity: In this scenario the bottleneck link capacity varies between 600 *kbps* and 2800 *kbps*.The aim of the test is to evaluate the reactivity and convergence of the algorithms. Our results show that SCReAM and FRACTaL react to the capacity changes and both the algorithms are initially conservative when ramping up to the bottleneck capacity and are otherwise aggressive.The average one-way delay in the case of SCReAM is higher because it is slow in detecting upcoming congestion.Therefore, we observe that SCReAM has a higher loss rate. FRACTaL sees fewer effective losses because, with FEC, it is able to recover packets lost when probing the network. SCReAM's average goodput is higher than the plain media rate of FRACTaL, but its combined media and FEC yields a similar bandwidth. Figure 5 shows a sample measurement when the forward path delay is set to 50ms. SCReAM keeps the media rate closer to the bottleneck link capacity, meanwhile FRACTaL repeatedly probes the link capacity by adding FEC and then undershooting when it senses congestion.The plots for queuing delay show that FRACTaL delay spikes are lower in amplitude, due to its undershooting process. Table 1 shows the average and the standard deviation for both algorithms.

RMCAT2—Multiple RTPs and Variable Capacity: This scenario aims to measure the fairness of the congestion control algorithm when two RTP streams are sharing the bottleneck link, the capacity of which is also varying over time. Bothflows start

			FCC	SCReAM
50ms	Flow 1	GP [kbps]	365.73 ± 28.28	524.64 ± 44.07
		LR [%]	0.17 ± 0.07	1.02 ± 0.51
		OWD [ms]	71.68 ± 3.14	87.93 ± 5.65
	Flow 2	GP [kbps]	327.97 ± 29.67	476.21 ± 40.65
		LR [%]	0.24 ± 0.09	0.94 ± 0.46
		OWD [ms]	70.16 ± 2.88	84.96 ± 4.78
300ms	Flow 1	GP [kbps]	391.44 ± 28.68	425.85 ± 45.53
		LR [%]	0.82 ± 0.13	6.76 ± 14.16
		OWD [ms]	341.85 ± 3.54	354.94 ± 24.64
	Flow 2	GP [kbps]	391.52 ± 19.48	440.38 ± 48.24
		LR [%]	0.76 ± 0.13	2.81 ± 2.43
		OWD [ms]	343.47 ± 2.79	352.83 ± 19.4

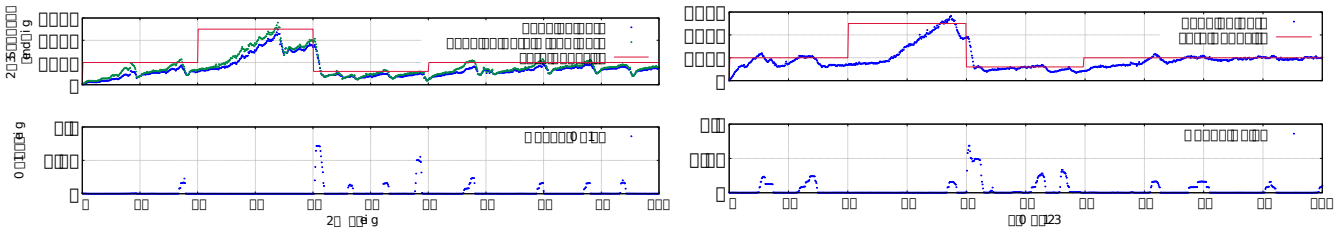
Table 2: Summary of RMCAT2 measurements. The FEC rate for 50ms, and 300ms: Flow 1) 77.03±3.99 kbps, 55.09±3.5 kbps; Flow 2) 75.05 ± 3.8 kbps, 54.82 ± 3.26 kbps. In all the cases the *FFRE* is 0.

simultaneously and have their paths have the same RTT. Figure 6 shows a sample simulation result for this scenario. Both algorithms converge and share the link fairly with each other. SCReAM reduces its media rate drastically when it encounters congestion. FRACTaL, on the other hand, performs better because it detects congestion quickly and responds by undershooting the media bitrate when it observes heavy congestion. Table 2 shows the average and standard deviation of OWD, Goodput, and loss rate for both algorithms. Similar to the previous scenario, we observed that SCReAM produces a higher effective loss rate than FRACTaL.

RMCAT3—Congested Feedback Path: In this test case, bidirectional media streams are used over a bottleneck link with time-varying capacity.Thus, RTP media and RTCP feedback share the same link and congestion impacts also the feedback.This allows evaluating algorithm performance with impaired feedback. We find that FRACTaL is more conservative than SCReAM with lower media rate and lower loss rates (not shown).

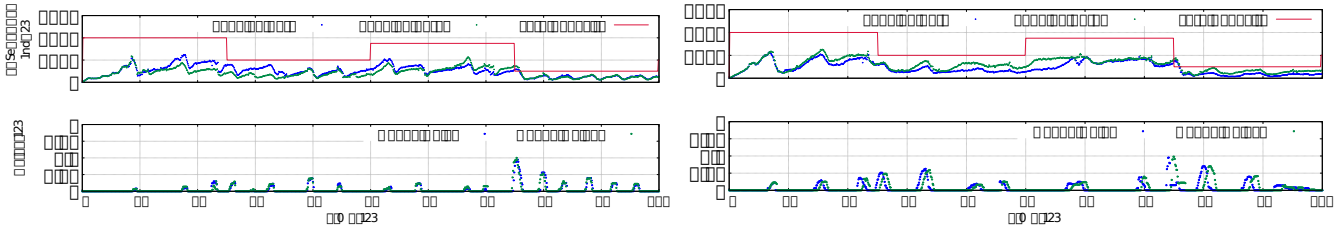
RMCAT4—Self-fairness: In this test case, multiplefl ows share the same bottleneck link whose capacity remains constant.The RTT is the same for eachfl ow, but they start at different times. Figure 7 shows a measurement for this scenario with a 50ms forward path delay. Both FRACTaL and SCReAM converge to a fair share ratio, although in case of FRACTaL the estimation causes faster convergence and lower loss rate. Table 3 shows the average and standard deviation of OWD, Goodput and loss rate for both algorithm. SCReAM utilizes the bandwidth better than FRACTaL especially forfl ow 1.This is due to the fact that SCReAM uses all the bits of the estimated bandwidth for the media bitrate, meanwhile FRACTaL probes with FEC packets before it decides to increase the actual media bitrate. While this slows down the media rate ramp, it improves the media quality because it is more stable over a longer period of time compared to SCReAM.

RMCAT5—Round Trip Time Fairness: This scenario aims to evaluate the congestion control algorithm when thefl ows share a common bottleneck link, but the RTTs and start times for the flows are different. A sample measurement for eachfl ow is shown in Figure 8; we observe that the performance of FRACTaL and SCReAM differ. FRACTaL converges better than SCReAM, whose rate sees oscillations.This is due to the feedback frequency, i.e., when the RTT is short, the feedback messages are sent more often, which results in SCReAM increasing its *cwnd* and media bitrate faster compared to the streams with longer RTTs.



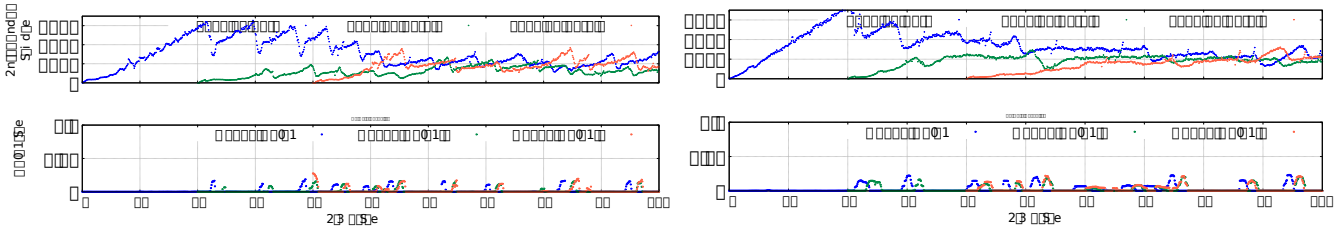
(a) FRACTaL, OWD = 50ms (b) SCReAM, OWD = 50ms

Figure 5: Performance of a single RTP stream in a scenario with varying link capacity (RMCAT1).



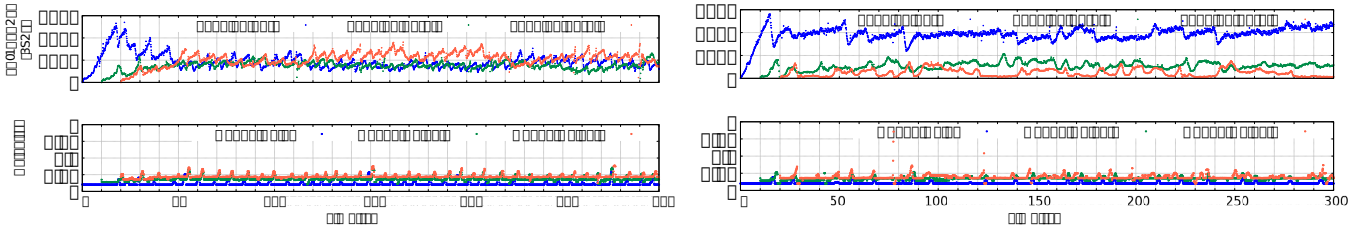
(a) FRACTaL (b) SCReAM

Figure 6: Performance of a two RTP streams competing in a scenario with varying link capacity (RMCAT2).



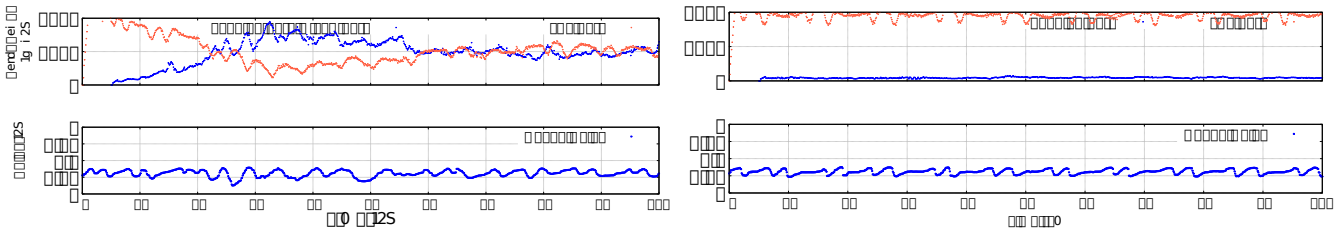
(a) FRACTaL (b) SCReAM

Figure 7: Performance of a three RTP streams, each with a different start time (RMCAT4).



(a) FRACTaL (b) SCReAM

Figure 8: Performance of a three RTP streams competing on a shared bottleneck, with different RTT for each (RMCAT5).



(a) FRACTaL (b) SCReAM

Figure 9: Performance of an RTP stream competing with a long-term TCP flow on a shared bottleneck (RMCAT6).

			FCC	SCReAM
50ms	Flow 1	GP [kbps]	1086.5 ± 143.06	1470.6 ± 233.69
		LR [%]	0 ± 0	0.26 ± 0.48
		OWD [ms]	55.47 ± 1.27	64.93 ± 3.2
	Flow 2	GP [kbps]	744.48 ± 243.82	851.98 ± 125.27
		LR [%]	0 ± 0	0.64 ± 0.97
		OWD [ms]	55.92 ± 1.59	66.93 ± 4.89
	Flow 3	GP [kbps]	645.53 ± 100.8	842.77 ± 112.68
		LR [%]	0 ± 0	0.09 ± 0.11
		OWD [ms]	56.37 ± 1.61	66.14 ± 3.23
300ms	Flow 1	GP [kbps]	1483.99 ± 115.78	1608.97 ± 128.05
		LR [%]	0 ± 0	0 ± 0.01
		OWD [ms]	311.59 ± 1.8	315.35 ± 3.29
	Flow 2	GP [kbps]	658.94 ± 58.62	717.26 ± 112.41
		LR [%]	0 ± 0	0.2 ± 0.23
		OWD [ms]	313.44 ± 2.65	315.35 ± 3.56
	Flow 3	GP [kbps]	645.3 ± 120.09	743.27 ±166.9
		LR [%]	0 ± 0	0.43 ± 0.86
		OWD [ms]	315.47 ± 2.94	316.99 ± 2.54

Table 3: Summary of RMCAT4 measurements. FEC rate for 50ms and 300**ms: Flow 1)** 127.15 ± 12.06 **kbps,** 129.83 ± 8.2 **kbps; Flow 2)** 101.37±16.81 **kbps,** 66.53±7.62 **kbps; Flow 3)** 92.84±9.28 **kbps,** 65.39 ± 10.27 **kbps; In all the cases the** *FFRE* **is 0.**

		FCC	SCReAM
Flow 1	GP [kbps]	547.71 ± 153.8	1971.6 ± 80.85
	LR [%]	0 ± 0	0 ± 0
	OWD [ms]	103.68 ± 46.43	81.99 ± 26.81
Flow 2	GP [kbps]	731.31 ± 157.39	440.09 ± 112.56
	LR [%]	0.04 ± 0.06	1.54 ± 1.46
	OWD [ms]	171.64 ± 56.42	142.67 ± 29.71
Flow 3	GP [kbps]	1161.45 ± 191.75	189.28 ± 55.16
	LR [%]	0.06 ± 0.12	4.49 ± 2.48
	OWD [ms]	217.28 ± 50.12	173.64 ± 25.82

Table 4: Summary of RMCAT5 measurements. FEC rate for thefl ows is: Flow 1) 91.13 ± 8.85 **kbps; Flow 2)** 75.92 ± 12.52 **kbps; Flow 3)** 98.78 ± 19.82 **kbps; In all the cases the** *FFRE* **is 0.**

FRACTaL probes and increases the media bitrate in separate time intervals, which introduces extra stability.Th is alternating mode makes FRACTaL more robust against RTT (or feedback frequency) variation because the algorithm senses upcoming congestion quicker. When media sources with different RTT compete, FRACTaL creates a combination of two effects: (1) the mediafl ow with larger RTT ramps up slower due to longer T_{valid}. But (2) it reacts slower to upcoming congestion and undershoots later.The latter makes the larger RTTfl ow less reactive and forces the media flow with shorter RTT to reduce its sending rate by a larger portion. Effect (1) favors the mediafl ow with shorter RTT while (2) favors the mediafl ow with longer RTT, which balance overall.

RMCAT6—Competing with a Long TCP Flow: In this scenario, we evaluate the performance of the congestion control algorithm when it competes with one long-term TCPfl ow on a shared bottleneck link.Th e RTTs are the same but the RTP stream starts a er the TCPfl ow. Figure 9 shows a sample measurement, in which FRACTaL is fairly competing with TCP, while SCReAM remains close to its minimal sending rate (the default is 64 kbps). As FRACTaL uses FEC for probing, this additional redundant media traffic pushes against the TCP traffic and causes TCP to reduce its sending rate. FEC also protect the mediafl ow against packet loss. *FFRE* is 6 − 12%.Th is scenario shows that FEC packets are indeed useful, especially for competing with TCP in order to reach a fair share.

		FCC	SCReAM
50ms	GP [kbps]	1045.86 ± 80.41	358.56 ± 617.13
	LR [%]	0.9 ± 0.23	13.24 ± 8.06
	OWD [ms]	348.64 ± 6.81	264.83 ± 95.02
	TFS	66.75 ± 6.69	112.05 ± 63.25
300ms	GP [kbps]	886.31 ± 59.16	39.47 ± 10.2
	LR [%]	1.24 ± 0.45	48.73 ± 13.16
	OWD [ms]	573.05 ± 14.97	508.8 ± 118.08
	TFS	79.93 ± 5.05	143.42 ± 1.41

Table 5: Summary of the measurements of RMCAT6. FEC rate and *FFRE* **for** 50**ms, and** 300**ms: Flow 1)** 79.98.64 ± 11.43 **kbps,** 6.59% ± 2.29**; Flow 2)** 71.07 ± 16.87 **kbps,** 12.22% ± 10.32**; Flow 3)** 77.07 ± 7.73 **kbps,** 12.74% ± 5.57**;**

		FCC	SCReAM
50ms	GP [kbps]	1121.75 ± 69.34	1128.56 ± 368.76
	LR [%]	0.27 ± 0.18	1.24 ± 0.9
	OWD [ms]	204.72 ± 55.47	132.27 ± 40.81
300ms	GP [kbps]	1240.78 ± 106.16	1065.13 ± 442.44
	LR [%]	0.58 ± 0.3	2.97 ± 3.41
	OWD [ms]	457.27 ± 21.08	357.95 ± 38.86

Table 6: Summary of the measurements of RMCAT6. FEC rate and *FFRE* **for** 50**ms and** 300**ms: Flow 1)** 119.64 ± 9.41**kbit/s,** 13.64% ± 19.58**; Flow 2)** 117.26 ± 10.9**kbit/s,** 4.66% ± 5.29**; Flow 3)** 107.16 ± 6.12**kbit/s,** 9.75% ± 9.5**;**

RMCAT7—Competing with Short TCP Flows: In this scenario, one RTP stream competes with several short term TCPfl ows on a shared bottleneck.Th e start times of the TCPfl ows are randomly selected and so is the amount of data to be fetched (typically 50–1500 KB). Our evaluation shows that the FRACTaL ramps up and the TCPfl ows ramp down to their respective fair share. In contrast, SCReAM remains at the minimal sending bitrate while the TCPfl ow is present and ramps up only when the TCPfl ows disappear.

6 CONCLUSION

We presented FRACTaL, a congestion control algorithm that incorporates FEC both for error resilience and for probing for available capacity. FRACTaL builds on top of our previous work, FBRA, and is able to eliminate some of FBRA's shortcomings, specifically its conservative ramp-up and not being able to compete well against TCP flows. FRACTaL achieves this by three measures: 1) Its FEC adapts to observed loss patterns in contrast to afi xed error protection scheme. 2) FRACTaL congestion control uses different congestion cues compared to FBRA: bytes infl ight and two sliding windows of 5s and 30s for its measurements. 3) FRACTaL modified the state machine in FBRA, which as a result improved reliability (much better frame recovery) and made the congestion control more stable.

We compare the performance of FRACTaL with SCReAM based on the standard RMCAT test cases in a testbed with the complete pipeline, i.e., with video codecs and reference videos, appropriate for real-time interactive communication.Th e performance evaluation was carried out in a test-bed with these algorithms implemented in OpenWebRTC framework. Our evaluation shows that FRACTaL has comparable goodput, better loss rate and lower network queue delay. Furthermore, we show that FRACTaL competes well with short- and long- TCP flows. Meanwhile, SCReAM when competing

with TCP collapses, which makes them not-ideal for deployment on the Internet, where TCP and UDP traffic share the same queues.

The RCMAT test cases offer useful reference scenarios for small scale comparison. In our future work, we will expand our evaluation to cover larger scale scenarios as well as paths across the Internet, and we will embrace measurement setups with wireless nodes. We also plan applying the FRACTaL to multipath scenarios in conjunction with MPRTP [34] to enhance the robustness of the latter, which we expect to benefit especially mobile and wireless scenarios.

REFERENCES

[1] H. Alvestrand. 2016. Overview: Real Time Protocols for Browser-based Applications. (2016), 22 pages. https://tools.ietf.org/pdf/dra-ietf-rtcweb-overview-18.pdf

[2] Jim Bankoski, Paul Wilkins, and Yaowu Xu. 2011. Technical overview of VP8, an open source video codec for the web. In *Proceedings - IEEE International Conference on Multimedia and Expo*. IEEE. https://doi.org/10.1109/ICME.2011.6012227

[3] R. Brandenburg, K. Gross, W. Wu, F. Boronat, and M. Montagud. 2012. *RTCP XR Report Block for One Way Delay metric Reporting*. Internet-Dra dra -wu-xrblock-rtcp-xr-one-way-delay-02.

[4] Lukasz Budzisz, Rade Stanojević, Arieh Schlote, Fred Baker, and Robert Shorten. 2011. On the fair coexistence of loss-and delay-based tcp. *IEEE/ACM Transactions on Networking* 19, 6 (2011), 1811–1824. https://doi.org/10.1109/TNET.2011.2159736

[5] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2016. Analysis and design of the google congestion control for web real-time communication (WebRTC). In *Proceedings of the 7th International Conference on Multimedia Systems - MMSys '16*. ACM Press, New York, New York, USA, 1–12. https://doi.org/10.1145/2910017.2910605

[6] Stefano D'Aronco, Laura Toni, Sergio Mena, Xiaoqing Zhu, and Pascal Frossard. 2017. Improved Utility-Based Congestion Control for Delay-Constrained Communication. *IEEE/ACM Transactions on Networking* 25, 1 (2017), 349–362. https://doi.org/10.1109/TNET.2016.2587579

[7] Jegadish Devadoss, Varun Singh, Jörg Ott, Chenghao Liu, Ye Kui Wang, and Igor Curcio. 2008. Evaluation of error resilience mechanisms for 3G conversational video. In *Proceedings - 10th IEEE International Symposium on Multimedia, ISM 2008*. IEEE, 378–383. https://doi.org/10.1109/ISM.2008.106

[8] S Floyd, M Handley, J Padhye, and J Widmer. 2000. Equation-Based Congestion Control for Unicast Applications. *Acm* 30, 4 (2000), 43–56. https://doi.org/10.1145/347057.347397

[9] T. Friedman, R. Caceres, and A. Clark. 2003. RTP Control Protocol Extended Reports (RTCP XR). RFC 3611 (Proposed Standard). (Nov. 2003), 55 pages. https://doi.org/10.17487/RFC3611

[10] L. Gharai. 2007. *RTP with TCP Friendly Rate Control*. Internet-Dra dra -ietf-avt-tfrc-profile-10.

[11] Stephen Hemminger. 2005. Network Emulation with NetEm. *Proceedings of the 6th Australian National Linux Conference (LCA 2005)* April (2005), 1–9.

[12] S. Holmer and H. Alvestrand. 2012. *A Google Congestion Control Algorithm for Real-Time Communication on the World Wide Web*. Internet-Dra dra -alvestrand-rtcweb-congestion-03.

[13] Stefan Holmer, Mikhal Shemer, and Marco Paniconi. 2013. Handling packet loss in WebRTC. In *2013 IEEE International Conference on Image Processing, ICIP 2013 - Proceedings*. IEEE, 1860–1864. https://doi.org/10.1109/ICIP.2013.6738383

[14] R. Huang and V. Singh. 2015. RTP Control Protocol (RTCP) Extended Report (XR) for Post-Repair Loss Count Metrics. RFC 7509 (Proposed Standard). (May 2015), 11 pages. https://doi.org/10.17487/RFC7509

[15] IETF. 2017. RMCAT working group of the IETF. (2017). http://datatracker.ietf.org/wg/rmcat

[16] IETF. Online. RTCWEB working group of the IETF. (Online). http://datatracker.ietf.org/wg/rtcweb

[17] ITU-T. 2003. G.114 One-way transmission time. *SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS International telephone connections and circuits fi General Recommendations on the transmission quality for an entire international telephone connection* (2003), 1–20.

[18] R. Jesup. 2013. *Congestion Control Requirements For RMCAT*. Internet-Dra dra -jesup-rmcat-reqs-01.

[19] I. Johansson and Z. Sarker. 2015. *Self-Clocked Rate Adaptation for Multimedia*. Internet-Dra dra -johansson-rmcat-scream-cc-05.

[20] Marcin Nagy, Varun Singh, Jörg Ott, and Lars Eggert. 2014. Congestion control using FEC for conversational multimedia communication. In *Proceedings of the 5th ACM Multimedia Systems Conference on - MMSys '14*. ACM Press, New York, New York, USA, 191–202. https://doi.org/10.1145/2557642.2557649

[21] B. Ngamwongwattana and R.Th ompson. 2010. Sync & Sense: VoIP Measurement Methodology for Assessing One-Way Delay Without Clock Synchronization. *IEEE Transactions on Instrumentation and Measurement* 59, 5 (may 2010), 1318–1326. https://doi.org/10.1109/TIM.2010.2043978

[22] P. O'Hanlon and K. Carlberg. 2013. *Congestion control algorithm for lower latency and lower loss media transport*. Internet-Dra dra -ohanlon-rmcat-dfl ow-02.

[23] Open Web RTC. 2017. Home Page of Open Web RTC. (2017). https://www.openwebrtc.org/

[24] J. Ott, V. Singh, and I. Curcio. 2014. RTP Control Protocol (RTCP) Extended Report (XR) for RLE of Discarded Packets. RFC 7097 (Proposed Standard). (Jan. 2014), 11 pages. https://doi.org/10.17487/RFC7097

[25] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. 2006. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). RFC 4585 (Proposed Standard). (July 2006), 51 pages. https://doi.org/10.17487/RFC4585 Updated by RFCs 5506, 8108.

[26] C. Perkins. 2016. *RTP Control Protocol (RTCP) Feedback for Congestion Control in Interactive Multimedia Conferences*. Internet-Dra dra -ietf-rmcat-rtp-cc-feedback-03.

[27] C. Perkins and O. Hodson. 1998. Options for Repair of Streaming Media. RFC 2354 (Informational). (June 1998), 12 pages. https://doi.org/10.17487/RFC2354

[28] C. Perkins and V. Singh. 2017. Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions. RFC 8083 (Proposed Standard). (March 2017), 25 pages. https://doi.org/10.17487/RFC8083

[29] R Rejaie, M Handley, and D Estrin. 1999. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In *Proc. of INFOCOM*. IEEE.

[30] Ericsson Research. 2017. OpenWebRTC-specific GStreamer plugins. (2017). https://github.com/EricssonResearch/openwebrtc-gst-plugins

[31] Z. Sarker, V. Singh, X. Zhu, and M. Ramalho. 2017. *Test Cases for Evaluating RMCAT Proposals*. Internet-Dra dra -ietf-rmcat-eval-test-05.

[32] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. 2003. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Internet Standard). (July 2003), 104 pages. https://doi.org/10.17487/RFC3550 Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164, 8083, 8108.

[33] Varun Singh. 2015. Protocols and Algorithms for Adaptive Multimedia Systems. (2015), 134 + app. 86 pages. http://urn.fi/URN:ISBN:978-952-60-6221-1

[34] Varun Singh, Saba Ahsan, and Jrg Ott. 2013. MPRTP: Multipath Considerations for Real-time Media. In *Proc. of ACM Multimedia Systems*.

[35] V. Singh, A. Begen, M. Zanaty, and G. Mandyam. 2017. *RTP Payload Format for Flexible Forward Error Correction (FEC)*. Internet-Dra dra -ietf-payload-fl exible-fec-scheme-05.

[36] Varun Singh, Albert Abello Lozano, and Jörg Ott. 2013. Performance analysis of receive-side real-time congestion control for WebRTC. In *Packet Video Workshop (PV), 2013 20th International*. IEEE, 1–8.

[37] V. Singh, S. Mc(istin, M. Ellis, and C. Perkins. 2013. Circuit Breakers for Multimedia Congestion Control. In *2013 20th International Packet Video Workshop*. 1–8. https://doi.org/10.1109/PV.2013.6691439

[38] V. Singh, M. Nagy, J. Ott, and L. Eggert. 2016. *Congestion Control Using FEC for Conversational Media*. Internet-Dra dra -singh-rmcat-adaptive-fec-03.

[39] Varun Singh, Jörg Ott, and Igor Curcio. 2009. Rate adaptation for conversational {3G} Video. In *Proc. of INFOCOM Workshop on MoViD*. IEEE, Brazil.

[40] V. Singh, J. Ott, and I. Curcio. 2014. RTP Control Protocol (RTCP) Extended Report (XR) Block for the Bytes Discarded Metric. RFC 7243 (Proposed Standard). (May 2014), 12 pages. https://doi.org/10.17487/RFC7243

[41] Varun Singh, Jörg Ott, and Igor D. D. Curcio. 2012. Rate-control for conversational video communication in heterogeneous networks. In *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 1–7. https://doi.org/10.1109/WoWMoM.2012.6263800

[42] Varun Singh, Jörg Ott, and Colin Perkins. 2012. Congestion control for interactive media: control loops & APIs. In *IAB/IRTF Workshop on Congestion Control for Interactive Real-Time Communication*.

[43] W3C. 2017. Web Real-Time Communications Working Group. (2017). http://www.w3.org/2015/06/webrtc-charter.html

[44] Yao Wang, S Wenger, Jiantao Wen, and a K Katsaggelos. 2000. Error resilient video coding techniques. *Signal Processing Magazine, IEEE* 17, 4 (2000), 61–82. https://doi.org/10.1109/79.855913

[45] Xiph.org. 2017. Video test media collection. (2017). http://media.xiph.org/video/derf/

[46] X. Zhu, R. Pan, M. Ramalho, S. Cruz, P. Jones, J. Fu, and S. D'Aronco. 2017. *NADA: A Unified Congestion Control Scheme for Real-Time Media*. Internet-Dra dra -ietf-rmcat-nada-04.

[47] Michael Zink, Oliver Künzel, Jens Schmitt, and Ralf Steinmetz. 2003. Subjective Impression of Variations in Layer Encoded Videos. In *Proceedings of the International Workshop onQu ality of Service*. Springer, 137–154. https://doi.org/10.1007/3-540-44884-5_8