Internet Engineering Task Force RMT WG
INTERNET-DRAFT
draft-ietf-rmt-bb-tfmcc-01.ps
Joerg Widmer/Univ. Mannheim
Mark Handley/ICIR

1 November 2002
Expires: May 2003

TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification

Status of this Document

This document is a product of the IETF RMT WG. Comments should be addressed to the authors, or the WG's mailing list at rmt@lbl.gov.

Abstract

This document specifies TCP-Friendly Multicast Congestion Control (TFMCC). TFMCC is a congestion control mechanism for multicast transmissions in a best-effort Internet environment. It is a single-rate congestion control scheme, where the sending rate is adapted to the receiver experiencing the worst network conditions. TFMCC is reasonably fair when competing for bandwidth with TCP flows and has a relatively low variation of throughput over time, making it suitable for applications such as streaming media where a relatively smooth sending rate is of importance.

Table of Contents

1. Introduction

This document specifies TCP-Friendly Multicast Congestion Control (TFMCC) [11]. TFMCC is a source-based, single-rate congestion control scheme that builds upon the unicast TCP-Friendly Rate Control mechanism (TFRC) [2]. TFMCC is stable and responsive under a wide range of network conditions and scales to receivers sets on the order of several thousand receivers. To support scalability, as much congestion control functionality as possible is located at the receivers. Each receiver continuously determines a desired receive rate that is TCP-friendly for the path from the sender to this receiver. Selected receivers then report the rate to the sender in feedback packets.

TFMCC is a building block as defined in RFC3048. Instead of specifying a complete protocol, this document simply specifies a congestion control mechanism that could be used in a transport protocol such as RTP [8], in an application incorporating end-to-end congestion control at the application level. This document does not discuss packet formats, reliability, or implementation-related issues.

TFMCC is designed to be reasonably fair when competing for bandwidth with TCP flows. A multicast flow is ‘‘reasonably fair’’ if its sending rate is generally within a factor of two of the sending rate of a TCP flow from the sender to the slowest receiver of the multicast group under the same network conditions.

In general, TFMCC has a low variation of throughput, which makes it suitable for applications such as streaming media where a relatively smooth sending rate is of importance. The penalty of having smooth throughput while competing fairly for bandwidth is a reduced responsiveness to changes in available bandwidth. Thus TFMCC should be used when the application has a requirement for smooth throughput, in particular, avoiding halving of the sending rate in response to a single packet drop. For applications that simply need to multicast as much data as possible in as short a time as possible, PGMCC [7] may be more suitable.

TFMCC is designed for applications that use a fixed packet size, and vary their sending rate in packets per second in response to congestion. Some audio applications require a fixed interval of time between packets and vary their packet size instead of their packet rate in response to congestion. The congestion control mechanism in this document cannot be used by those applications.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 and indicate requirement levels for compliant TFMCC implementations.

1.2. Related Documents

As described in RFC3048, TFMCC is a building block that is intended to be used, in conjunction with other building blocks, to help specify a protocol instantiation. In particular, TFMCC is a suitable congestion control building block for NACK-Oriented Reliable Multicast (NORM) [1].

1.3. Environmental Requirements and Considerations

TFMCC is intended to be a congestion control scheme that can be used in a complete protocol instantiation that delivers objects and streams (both reliable content delivery and streaming of multimedia information).

TFMCC is most applicable for sessions that deliver a substantial amount of data, i.e., in length from hundreds of kilobytes to many gigabytes, and whose duration is in the order of tens of seconds or more.

TFMCC is intended for multicast delivery. There are currently two models of multicast delivery, the Any-Source Multicast (ASM) model as defined in RFC1112 and the Source-Specific Multicast (SSM) model as defined in [3]. TFMCC works with both multicast models, but in a slightly different way. When using ASM, feedback from the receivers is multicast to the sender as well as to all other receivers. Feedback can be either multicast on the same group address used for sending data or on a separate multicast feedback group address. For SSM, the receivers must unicast the feedback directly to the sender. Hence, feedback from a receiver will not be received by other receivers.

TFMCC inherently works with all types of networks, including LANs, WANs, Intranets, the Internet, asymmetric networks, wireless networks, and satellite networks. However, in some network environments varying the sending rate to the receivers may not be advantageous (e.g., for a satellite or wireless network, there may be no mechanism for receivers to effectively reduce their reception rate since there may be a fixed transmission rate allocated to the session).

2. Protocol Overview

TFMCC extends the basic mechanisms of TFRC into the multicast domain. In order to compete fairly with TCP, TFMCC receivers individually measure the prevalent network conditions and calculate a rate that is TCP-friendly on the path from the sender to themselves. The rate is determined using an equation for TCP throughput, which roughly describes TCP's sending rate as a function of the loss event rate, round-trip time (RTT), and packet size. We define a loss event as one or more lost or marked packets from the packets received during one RTT, where a marked packet refers to a congestion indication from Explicit Congestion Notification (ECN) [6]. The sending rate of the multicast transmission is adapted to the receiver experiencing the worst network conditions.

Basically, TFMCC's congestion control mechanism works as follows:

- Each receiver measures the loss event rate and its RTT to the sender.

- Each receiver then uses this information, together with an equation for TCP throughput, to derive a TCP-friendly sending rate.
- Through a distributed feedback suppression mechanism, only a subset of the receivers are allowed to give feedback to prevent a feedback implosion at the sender. The feedback mechanism ensures that receivers reporting a low desired transmission rate have a high probability of sending feedback.
- Receivers whose feedback is not suppressed report the calculated transmission rate back to the sender in so-called receiver reports. The receiver reports serve two purposes: they inform the sender about the appropriate transmit rate, and they allow the receivers to measure their RTT.
- The sender selects the receiver that reports the lowest rate as current limiting receiver (CLR). Whenever feedback with an even lower rate reaches the sender, the corresponding receiver becomes CLR and the sending rate is reduced to match that receiver's calculated rate. The sending rate increases when the CLR reports a calculated rate higher than the current sending rate.

The dynamics of TFMCC are sensitive to how the measurements are performed and applied and what feedback suppression mechanism is chosen. We recommend specific mechanisms below to perform and apply these measurements. Other mechanisms are possible, but it is important to understand how the interactions between mechanisms affect the dynamics of TFMCC.

2.1. TCP Throughput Equation

Any realistic equation giving TCP throughput as a function of loss event rate and RTT should be suitable for use in TFMCC. However, we note that the TCP throughput equation used must reflect TCP's retransmit timeout behavior, as this dominates TCP throughput at higher loss rates. We also note that the assumptions implicit in the throughput equation about the loss event rate parameter have to be a reasonable match to how the loss rate or loss event rate is actually measured. While this match is not perfect for the throughput equation and loss rate measurement mechanisms given below, in practice the assumptions turn out to be close enough.

The throughput equation we currently recommend for TFMCC is a slightly simplified version of the throughput equation for Reno TCP from [4]:

$$X = \frac{8s}{R(\sqrt{\frac{2p}{3}} + (12\sqrt{\frac{3p}{8}})p(1+32p^2))} \qquad (1)$$

where

X is the transmit rate in bits/second.

s is the packet size in bytes.

R is the round-trip time in seconds.

p is the loss event rate, between 0.0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted.

In future, different TCP equations may be substituted for this equation. The requirement is that the throughput equation be a reasonable approximation of the sending rate of TCP for conformant TCP congestion control.

The parameters s (packet size), p (loss event rate) and R (RTT) need to be measured or calculated by a TFMCC implementation. The measurement of R is specified in Section 4.3.2, and the measurement of p is specified in Section 5. The parameter s (packet size) is normally known to an application. This may not be so in two cases:

- The packet size naturally varies depending on the data. In this case, although the packet size varies, that variation is not coupled to the transmit rate. It should normally be safe to use an estimate of the mean packet size for s.
- The application needs to change the packet size rather than the number of packets per second to perform congestion control. This would normally be the case with packet audio applications where a fixed interval of time needs to be represented by each packet. Such applications need to have a different way of measuring parameters.

Currently, TFMCC cannot be used for the second class of applications.

2.2. Packet Contents

Before specifying the sender and receiver functionality, we describe the contents of the data packet headers sent by the sender and feedback packets sent by the receivers. As TFMCC will be used along with a transport protocol, we do not specify packet formats, since these depend on the details of the transport protocol used.

The recommended representation of the header fields is given below. Alternatively, if the computational overhead of a floating point representation is prohibitive, fixed point arithmetic can be used at the expense of larger packet headers.

2.2.1. Data Packets

Each packet sent by the data sender contains the following information:

- A sequence number i. This number is incremented by one for each data packet transmitted. The field must be sufficiently large that it does not wrap causing two different packets with the same sequence number to be in the receiver's recent packet history at the same time. In most cases the sequence number will be supplied by the transport protocol used along with TFMCC.
- A suppression rate X_supp in bits/s. Only receivers with a calculated rate lower than the suppression rate are eligible to give feedback, unless their RTT is higher than the maximum RTT described below in which case they are also eligible to give feedback. The suppression rate should be represented as a 12 bit floating point value with 5 bits for the unsigned exponent

and 7 bits for the unsigned mantissa (to represent rates from 100 bit/s to 400 Gbit/s with an error of less than 1%).

- A timestamp `ts_i` indicating when the packet is sent. The resolution of the timestamp should typically be milliseconds and the timestamp should be an unsigned integer value no less than 16 bits wide.
- A receiver ID `r` and a copy of the timestamp `ts_r` of that receiver's last report, which allows the receiver to measure its RTT. The receiver ID is described in the next section. The resolution of the timestamp echo should be milliseconds and the timestamp should be an unsigned integer value no less than 16 bits wide.
- A flag `is_CLR` indicating whether the receiver with ID `r` is the CLR.
- A feedback round counter `fb_nr`. This counter is incremented by the sender at the beginning of a new feedback round to notify the receivers than all feedback for older rounds should be suppressed. The feedback round counter should be at least 4 bits wide.
- A maximum RTT value `R_max`, representing the maximum of the RTTs of all receivers. The RTT should be measured in milliseconds. An 8 bit floating point value with 4 bits for the unsigned exponent and 4 bits for the unsigned mantissa (to represent RTTs from 1 millisecond to 64 seconds with an error of ca. 6%) should be used for the representation.

2.2.2. Feedback Packets

TFMCC receivers use two different formats for feedback packets depending on whether they have made at least one RTT measurement or not. Each feedback packet sent by a data receiver contains the following information:

- A unique receiver ID `r`. In most cases the receiver ID will be supplied by the transport protocol, but it may simply be the IP address of the receiver.
- A flag `have_RTT` indicating whether the receiver has made at least one RTT measurement since it joined the session.
- A flag `have_loss` indicating whether the receiver experienced at least one loss event since it joined the session.
- A flag `receiver_leave` indicating that the receiver will leave the session (and should therefore not be CLR).
- A timestamp `ts_r` indicating when the feedback packet is sent. The representation of the timestamp should be the same as that of the timestamp echo in the data packets.
- An echo of the timestamp of the last data packet received. If the last packet received at the receiver has sequence number `i`, then `ts_i` is included in the feedback. If there is a delay `t_d` between receiving that last data packet and sending feedback, then `ts_i' = ts_i + t_d` is included in the feedback instead of `ts_i`. The representation of the timestamp echo should be the same as that of the timestamp in the data packets.
- A feedback round echo `fb_nr`, reflecting the highest feedback round counter value received so far. The representation of the feedback round echo should be the same as the one used for the

feedback round counter in the data packets.

- The desired sending rate `X_r`. This is the rate calculated by the receiver to be TCP-friendly on the path from the sender to this receiver. The representation of the desired sending rate should be the same as that of the suppression rate in the data packets.

3. Data Sender Protocol

The data sender multicasts a stream of data packets to the data receivers at a controlled rate. Whenever feedback is received, the sender checks if it is necessary to switch CLRs and to readjust the sending rate.

The main tasks that have to be provided by a TFMCC sender are:

- adjusting the sending rate,
- controlling receiver feedback, and
- assisting receiver-side RTT measurements.

3.1. Sender Initialization

At initialization of the sender, the maximum RTT is set to a value that should be larger than the highest RTT to any of the receivers. It should not be smaller than 500 milliseconds for operation in the public Internet. The sending rate `X` is initialized to 1 packet per maximum RTT.

3.2. Determining the Maximum RTT

For each feedback packet that arrives at the sender, the sender computes the instantaneous RTT to the receiver as

```
R_r = t_now - ts_i
```

where `t_now` is the time the feedback packet arrived. Receivers will have adjusted `ts_i` for the time interval between receiving the last data packet and sending the corresponding report so that this interval will not be included in `R_r`. If the instantaneous RTT is larger than the current maximum RTT, the maximum RTT is increased to that value

```
R_max = R_r
```

Otherwise, if no feedback with a higher instantaneous RTT than the maximum RTT is received during a feedback round (see Section 3.4), the maximum RTT is reduced exponentially to

```
R_max = R_max * 0.9
```

which results in a slow decrease over a number of feedback rounds.

The maximum RTT is mainly used for feedback suppression among receivers with heterogeneous RTTs. Feedback suppression is closely coupled to the sending of data packets and for this reason, the maximum RTT must not decrease below the maximum time interval between consecutive data packets:

```
R_max = max(R_max, s/X + t_gran)
```

where t_gran is the granularity of the sender's system clock (see Section 3.7).

3.3. Adjusting the Sending Rate

When a feedback packet from receiver r arrives at the sender, the sender has to check whether it is necessary to adjust the transmission rate and to switch to a new CLR.

How the rate is adjusted depends on the desired rate X_r of the receiver report. We distinguish four cases:

1 If no CLR is present, receiver r becomes the current limiting receiver. The sending rate X is directly set to X_r, so long as this would result in a rate increase of less than 8s/R_max bits/s. Otherwise X is gradually increased to X_r at an increase rate of no more than 8s/R_max bits/s every R_max seconds.

2 If receiver r is not the CLR but a CLR is present, then receiver r becomes the current limiting receiver if X_r is less than the current sending rate X and the receiver_leave flag of that receiver's report is not set. Furthermore, the sending rate is reduced to X_r.

3 If receiver r is not the CLR but a CLR is present and the receiver_leave flag of the CLR's last report was set, then receiver r becomes the current limiting receiver. However, if X_r > X, the sending rate is not increased to X_r for the duration of a feedback round to allow other (lower rate) receivers to give feedback and be selected as CLR.

4 If receiver r is the CLR, the sending rate is set to the minimum of X_r and X + 8s/R_max bits/s.

If the receiver has not yet measured its RTT (i.e., the have_RTT flag is set), the receiver report will include a desired rate that is based on the maximum RTT rather than the actual RTT to that receiver. In this case, the sender adjusts the desired rate using its measurement of the instantaneous RTT R_r to that receiver:

```
X_r' = X_r * R_max / R_r
```

X_r' is then used instead of X_r to detect whether to switch to a new CLR.

If the TFMCC sender receives no reports from the CLR for at least 10 RTTs, it assumes that the CLR crashed or left the group. A new CLR is selected from the feedback that subsequently arrives at the sender, and we increase as in case 3 above.

In the absence of *any* feedback from any of the receivers it is necessary to reduce the sending rate. For every 10 consecutive RTTs without feedback, the sending rate is cut in half. The rate is at most reduced to one packet every 64 seconds. Note that when receivers stop receiving data packets, they will stop sending feedback. This eventually causes the sending rate to be reduced in the case of network failure. If the network subsequently recovers, a linear increase to the calculated rate of the CLR will occur at 8s/R_max bits/s every R_max.

3.4. Controlling Receiver Feedback

The receivers allowed to send a receiver report are determined in so-called feedback rounds. Feedback rounds have a duration `T` of six times the maximum RTT. In case the multicast model is ASM, i.e., receiver feedback is multicast to the whole group, the duration of a feedback round may be reduced to four times the maximum RTT.

Only receivers wishing to report a rate that is lower than the suppression rate `X_supp`, or those with a higher RTT than `R_max` may send feedback. At the beginning of each feedback round, `X_supp` is set to the highest possible value that can be represented. When feedback arrives at the sender over the course of a feedback round, `X_supp` is decreased such that more and more feedback is suppressed towards the end of the round. How receiver feedback is spread out over the feedback round is discussed in Section 4.5.

Whenever non-CLR feedback for the current round arrives at the sender, `X_supp` is reduced to

```
X_supp = (1-g) * X_r
```

if `X_supp > X_r`. Feedback that causes the corresponding receiver to be selected as CLR, but was from a non-CLR receiver at the time of sending also contributes to the feedback suppression. Note that `X_r` must not be adjusted by the sender to reflect the receiver's real RTT in case `X_r` was calculated using the maximum RTT, as is done for setting the sending rate (Section 3.3), otherwise a feedback implosion is possible. The parameter `g` determines to what extent higher rate feedback can suppress lower rate feedback. This mechanism guarantees, that the lowest calculated rate reported lies within a factor of `g` of the actual lowest calculated rate of the receiver set (see [10]). A value of `g` of 0.1 is recommended.

After a time span of `T`, the feedback round ends if non-CLR feedback was received during that time. Otherwise, the feedback round ends as soon as the first non-CLR feedback meesage arrives at the sender but at most after `2T`. The feedback round counter is incremented by one and the suppression rate `X_supp` is reset to the highest representable value. The feedback round counter restarts with round 0 after a wrap-around.

3.5. Assisting Receiver-Side RTT Measurements

Receivers measure their RTT by sending a timestamp with a receiver report, which is echoed in the next data packet. Only one receiver ID and timestamp can be included in a data packet. If multiple feedback messages from different receivers arrive at the sender during the time interval between two data packets, the sender has to decide which receiver to allow to measure RTT.

The sender's timestamp echoes are prioritized in the following order:

1. a new CLR (after a change of CLR's) or a CLR without any previous RTT measurements
2. receivers without any previous RTT measurements in the order of the feedback round echo of the corresponding receiver report (i.e., older feedback first)
3. non-CLR receivers with previous RTT measurements, again in ascending order of the feedback round echo of the report

4. the CLR

Ties are broken in favor of the receiver with the lowest reported rate.

It is necessary to account for the time that elapses between receiving a report and sending the next data packet. This time needs to be deducted from the RTT and thus has to be added to the receiver's timestamp value. The receiver ID and the adjusted timestamp of the receiver with the highest priority are then included in the next data packet.

Whenever no feedback packets arrive in the interval between two data packets, the CLR's last timestamp, adjusted by the appropriate offset, is echoed. When the number of packets per RTT is so low that all packets carry a non-CLR receiver's timestamp, the CLR's timestamp and ID are included in a data packet at least once per feedback round.

3.6. Slowstart

TFMCC uses a slowstart mechanism to quickly approach its fair bandwidth share at the start of a session. During slowstart, the sending rate increases exponentially. The rate increase is limited to the minimum of the rates included in the receiver reports and receivers report twice the rate at which they currently receive data. As in normal congestion control mode, the receiver with the smallest reported rate becomes CLR. Since a receiver can never receive data at a rate higher than its link bandwidth, this effectively limits the overshoot to twice this bandwidth. In case the resulting increase over `R_max` is less than `8s/R_max` bits/s, the sender may choose to increase the rate by up to `8s/R_max` bits/s every `R_max`. The current sending rate is gradually adjusted to the target rate reported in the receiver reports over the course of a RTT. Slowstart is terminated as soon as any one of the receivers experiences its first packet loss. Since that receiver's calculated rate will be lower than the current sending rate, the receiver will be selected as CLR.

During slowstart, the upper bound on the rate increase of `8s/R_max` bits/s every RTT does not apply. Only after the TFMCC sender receives the first report with the `have_loss` flag set is the rate increase limited in this way.

3.7. Scheduling of Packet Transmissions

As TFMCC is rate-based, and as operating systems typically cannot schedule events precisely, it is necessary to be opportunistic about sending data packets so that the correct average rate is maintained despite the coarse-grain or irregular scheduling of the operating system. Thus a typical sending loop will calculate the correct inter-packet interval, `t_ipi`, as follows:

```
t_ipi = s/X
```

When a sender first starts sending at time `t_0`, it calculates `t_ipi`, and calculates a nominal send time `t_1 = t_0 + t_ipi` for packet 1. When the application becomes idle, it checks the current time, `t_now`, and then requests re-scheduling after `(t_ipi - (t_now - t_0))` seconds. When the application is re-scheduled, it checks the current time, `t_now`, again. If `(t_now > t_1 - delta)` then packet 1 is sent (see below for `delta`).

Now a new t_ipi may be calculated, and used to calculate a nominal send time t_2 for packet 2: t2 = t_1 + t_ipi. The process then repeats, with each successive packet's send time being calculated from the nominal send time of the previous packet.

In some cases, when the nominal send time, t_i, of the next packet is calculated, it may already be the case that t_now > t_i - delta. In such a case the packet should be sent immediately. Thus if the operating system has coarse timer granularity and the transmit rate is high, then TFMCC may send short bursts of several packets separated by intervals of the OS timer granularity.

The parameter delta is to allow a degree of flexibility in the send time of a packet. If the operating system has a scheduling timer granularity of t_gran seconds, then delta would typically be set to:

```
delta = min(t_ipi/2, t_gran/2)
```

t_gran is 10 milliseconds on many Unix systems. If t_gran is not known, a value of 10 milliseconds can be safely assumed.

4. Data Receiver Protocol

Receivers measure the current network conditions, namely RTT and loss event rate, and use this information to calculate a rate that is fair to competing traffic. The rate is then communicated to the sender in receiver reports. Due to the potentially large number of receivers, it is undesirable that all receivers send reports, especially not at the same time.

In the description of the receiver functionality, we will first address how the receivers measure the network parameters and then discuss the feedback process.

4.1. Receiver Initalization

The receiver is initialized when it receives the first data packet. The RTT is set to the maximum RTT value contained in the data packet. This initial value is used as the receiver's RTT until the first real RTT measurement is made. The loss event rate is initialized to 0.

4.2. Receiver Leave

A receiver that sends feedback but wishes to leave the TFMCC session within the next feedback round may indicate the pending leave by setting the receiver_leave flag in its report. If the leaving receiver is the CLR, the receiver_leave flag should be set for all the reports within the feedback round before the leave takes effect.

4.3. Measurement of the Network Conditions

Receivers have to update their estimate of the network parameters with each new data packet they receive.

4.3.1. Updating the Loss Event Rate

When a data packet is received, the receiver adds the packet to the packet history. It then recalculates the new value of the loss event rate `p`. The loss event rate measurement mechanism is described separately in Section 5.

4.3.2. Basic Round-Trip Time Measurement

When a receiver gets a data packet that carries the receiver's own ID in the `r` field, the receiver updates its RTT estimate.

1. The current RTT is calculated as:

   ```
   R_sample = t_now - ts_r
   ```

   where `t_now` is the time the data packet arrives at the receiver and `ts_r` is the receiver report timestamp echoed in the data packet.

2. The smoothed RTT estimate `R` is updated:

   ```
   If no feedback has been received before
       R = R_sample
   Else
       R = q*R + (1-q)*R_sample
   ```

   A filter parameter `q` of 0.5 is recommended for non-CLR receivers. The CLR performs RTT measurements much more frequently and hence should use a higher filter value. We recommend using `q=0.9`. Note that TFMCC is not sensitive to the precise value for the filter constant.

4.3.3. One-Way Delay Adjustments

When a RTT measurement is performed, the receiver also determines the one-way delay `D_r` from itself to the sender:

```
D_r = ts_r - ts_i
```

where `ts_i` and `ts_r` are the timestamp and timestamp echo contained in the data packet. With each new data packet `i'`, the receiver can now calculate an updated RTT estimate as:

```
R' = D_r + t_now - ts_i'
```

Inbetween RTT measurements, the updated `R'` is used instead of the smoothed RTT `R`. When a new measurement is made, all interim one-way delay measurements are discarded (i.e., the smoothed RTT is updated according to Section 4.3.2 without taking one-way delay adjustments into account).

For the one-way delay measurements, the clocks of sender and receivers need not be synchronized. Clock skew will cancel itself out when both one-way measurements are added to form a RTT estimate, as long as clock drift between real RTT measurements is negligible.

4.3.4. Receive Rate Measurements

When a receiver has not experienced any loss events, it cannot calculate a TCP-friendly rate to include in the receiver reports. Instead, the receiver measures the current receive rate and sets the desired rate `X_r` to twice the receive rate.

The receive rate in bits/s is measured as the number of bits received over the last `k` RTTs, taking into account the IP and transport packet headers, but excluding the link-layer packet headers. A value for `k` between 2 and 4 is recommended.

4.4. Setting the Desired Rate

When a receiver measures a non-zero loss event rate, it calculates the desired rate using Equation (1). In case no RTT measurement is available yet, the maximum RTT is used instead of the receiver's RTT. The desired rate is updated whenever the loss event rate or the RTT changes.

As mentioned above, calculation of the desired rate is not possible before the receiver experiences the first loss event and in that case twice the rate at which data is received is included in the receiver reports as `X_r`. This mechanism allows the sender to slowstart as described in Section 3.6.

4.5. Feedback and Feedback Suppression

Let `fb_nr` be the highest feedback round counter value received by a receiver. When a new data packet arrives with a higher feedback round counter than `fb_nr`, a new feedback round begins and `fb_nr` is updated. Outstanding feedback for the old round is cancelled. In case a feedback number with a value that is more than half the feedback number space lower than `fb_nr` is received, the receiver assumes that the feedback round counter wrapped and also cancels the feedback timer and updates `fb_nr`.

The CLR sends its feedback independently from all the other receivers once per RTT. Its feedback does not suppress other feedback and cannot be suppressed by other receiver's feedback.

Non-CLR receivers set a feedback timer at the beginning of a feedback round. Using an exponentially weighted random timer mechanism, the feedback timer is set to expire after

```
t = max(T * (1 + log(x)/log(N)), 0)
```

where

x is a random variable uniformly distributed in (0,1],

T is the duration of a feedback round,

N is an estimated upper bound on the number of receivers.

N is a constant specific to the TFMCC protocol. Since TFMCC scales to up to thousands of receivers, setting N to 10,000 for all receivers (and limiting the TFMCC session to at most 10,000 receivers) is recommended.

A feedback packet is sent when the feedback timer expires, unless the timer is cancelled beforehand. When the multicast model is ASM, feedback is multicast to the whole group, otherwise the feedback is unicast to the sender. The feedback packet includes the calculated rate valid at the time the feedback packet is sent (not the rate at the point of time when the feedback timer is set). The copy of the timestamp ts_i of the last data packet received, which is included in the feedback packet, needs to be adjusted by the time interval between receiving the data packet and sending the report to allow the sender to correctly infer the instantaneous RTT (i.e., that time interval has to be added to the timestamp value).

The timer is cancelled if a data packet with a lower suppression rate than the receiver's calculated rate and a higher or equal maximum RTT than the receiver's RTT is received. Likewise, a data packet indicating the beginning of a new feedback round cancels all feedback for older rounds. In case of ASM, the timer is also cancelled if a feedback packet from another non-CLR receiver reporting a lower rate is received.

The feedback suppression process is complicated by the fact that the calculated rates of the receivers will change during a feedback round. If the calculated rates decrease rapidly for all receivers, feedback suppression can no longer prevent a feedback implosion since earlier feedback will always report a higher rate than current feedback. To make the feedback suppression mechanism robust in the face of changing rates, it is necessary to introduce X_fbr, the calculated rate of a receiver at the beginning of a feedback round. A receiver needs to suppress its feedback not only when the suppression rate is less than the receiver's current calculated rate but also in the case that the suppression rate falls below X_fbr.

When the maximum RTT changes significantly during one feedback round, it is necessary to reschedule the feedback timer in proportion to the change.

```
t = t * R_max / R_max'
```

where R_max is the new maximum RTT and R_max' is the previous maximum RTT. The same considerations hold, when the last data packets were received more than a time interval of R_max ago. In this case, it is necessary to add the difference of the inter-packet gap and the maximum RTT to the feedback time to prevent a feedback implosion (e.g., in case the sender crashed).

```
t = t + max(t_now - tr_i - R_max, 0)
```

where tr_i is the time when the last data packet arrived at the receiver.

More details on the characteristics of the feedback suppression mechanism can be found in [10] and [11].

5. Calculation of the Loss Event Rate

Obtaining an accurate and stable measurement of the loss event rate is of primary importance for TFMCC. Loss rate measurement is performed at the receiver, based on the detection of lost or marked packets from the sequence numbers of arriving packets.

5.1. Detection of Lost or Marked Packets

TFMCC assumes that all packets contain a sequence number that is incremented by one for each packet that is sent. For the purposes of this specification, we require that if a lost packet is retransmitted, the retransmission is given a new sequence number that is the latest in the transmission sequence, and not the same sequence number as the packet that was lost. If a transport protocol has the requirement that it must retransmit with the original sequence number, then the transport protocol designer must figure out how to distinguish delayed from retransmitted packets and how to detect lost retransmissions.

The receivers each maintain a data structure that keeps track of which packets have arrived and which are missing. For the purposes of specification, we assume that the data structure consists of a list of packets that have arrived along with the timestamp when each packet was received. In practice this data structure will normally be stored in a more compact representation, but this is implementation-specific.

The loss of a packet is detected by the arrival of at least three packets with a higher sequence number than the lost packet. The requirement for three subsequent packets is the same as with TCP, and is to make TFMCC more robust in the presence of reordering. In contrast to TCP, if a packet arrives late (after 3 subsequent packets arrived) at a receiver, the late packet can fill the hole in the reception record, and the receiver can recalculate the loss event rate. Future versions of TFMCC might make the requirement for three subsequent packets adaptive based on experienced packet reordering, but we do not specify such a mechanism here.

For an ECN-capable connection, a marked packet is detected as a congestion event as soon as it arrives, without having to wait for the arrival of subsequent packets.

5.2. Translation from Loss History to Loss Events

TFMCC requires that the loss event rate be robust to several consecutive packets lost where those packets are part of the same loss event. This is similar to TCP, which (typically) only performs one halving of the congestion window during any single RTT. Thus the receivers needs to map the packet loss history into a loss event record, where a loss event is one or more packets lost in a RTT.

To determine whether a lost or marked packet should start a new loss event, or be counted as part of an existing loss event, we need to compare the sequence numbers and timestamps of the packets that arrived at the receiver. For a marked packet `S_new`, its reception time `T_new` can be noted directly. For a lost packet, we can interpolate to infer the nominal "arrival time". Assume:

S_loss is the sequence number of a lost packet.

S_before is the sequence number of the last packet to arrive with sequence number before S_loss.

S_after is the sequence number of the first packet to arrive with sequence number after S_loss.

T_before is the reception time of S_before.

T_after is the reception time of S_after.

Note that T_before can either be before or after T_after due to reordering.

For a lost packet S_loss, we can interpolate its nominal "arrival time" at the receiver from the arrival times of S_before and S_after. Thus

```
T_loss = T_before + ( (T_after - T_before)
                * (S_loss - S_before)/(S_after - S_before) );
```

Note that if the sequence space wrapped between S_before and S_after, then the sequence numbers must be modified to take this into account before performing the calculation. If the largest possible sequence number is S_max, and S_before > S_after, then modifying each sequence number S by S' = (S + (S_max + 1)/2) mod (S_max + 1) would normally be sufficient.

If the lost packet S_old was determined to have started the previous loss event, and we have just determined that S_new has been lost, then we interpolate the nominal arrival times of S_old and S_new, called T_old and T_new respectively.

If T_old + R >= T_new, then S_new is part of the existing loss event. Otherwise S_new is the first packet of a new loss event.

5.3. Inter-Loss Event Interval

If a loss interval, A, is determined to have started with packet sequence number S_A and the next loss interval, B, started with packet sequence number S_B, then the number of packets in loss interval A is given by (S_B - S_A).

5.4. Average Loss Interval

To calculate the loss event rate p, we first calculate the average loss interval. This is done using a filter that weights the n most recent loss event intervals in such a way that the measured loss event rate changes smoothly.

Weights w_0 to w_(n-1) are calculated as:

```
If (i < n/2)
   w_i = 1;
Else
   w_i = 1 - (i - (n/2 - 1))/(n/2 + 1);
```

Thus if n=8, the values of w_0 to w_7 are:

1.0, 1.0, 1.0, 1.0, 0.8, 0.6, 0.4, 0.2

The value n for the number of loss intervals used in calculating the loss event rate determines TFMCC's speed in responding to changes in the level of congestion. As currently specified, TFMCC should not be used for values of n significantly greater than 8, for traffic that might compete in the global Internet with TCP. At the very least, safe operation with values of n greater than 8 would require a slight change to TFMCC's mechanisms to include a more severe response to two or more round-trip times with heavy packet loss.

When calculating the average loss interval we need to decide whether to include the interval since the most recent packet loss event. We only do this if it is sufficiently large to increase the average loss interval.

Thus if the most recent loss intervals are I_0 to I_n, with I_0 being the interval since the most recent loss event, then we calculate the average loss interval I_mean as:

```
I_tot0 = 0;
I_tot1 = 0;
W_tot = 0;
for (i = 0 to n-1) {
  I_tot0 = I_tot0 + (I_i * w_i);
  W_tot = W_tot + w_i;
}
for (i = 1 to n) {
  I_tot1 = I_tot1 + (I_i * w_(i-1));
}
I_tot = max(I_tot0, I_tot1);
I_mean = I_tot/W_tot;
```

The loss event rate, p is simply:

```
p = 1 / I_mean;
```

5.5. History Discounting

As described in Section 5.4, the most recent loss interval is only assigned 4/(3*n) of the total weight in calculating the average loss interval, regardless of the size of the most recent loss interval. This section describes an optional history discounting mechanism that allows the TFMCC receivers to adjust the weights, concentrating more of the relative weight on the most recent loss interval, when the most recent loss interval is more than twice as large as the computed average loss interval.

To carry out history discounting, we associate a discount factor `DF_i` with each loss interval `L_i`, where each discount factor is a floating point number. The discount array maintains the cumulative history of discounting for each loss interval. At the beginning, the values of `DF_i` in the discount array are initialized to 1:

```
for (i = 1 to n) {
  DF_i = 1;
}
```

History discounting also uses a general discount factor `DF`, also a floating point number, that is also initialized to 1. First we show how the discount factors are used in calculating the average loss interval, and then we describe later in this section how the discount factors are modified over time.

As described in Section 5.4 the average loss interval is calculated using the `n` previous loss intervals `I_1, ..., I_n`, and the interval `I_0` that represents the number of packets received since the last loss event. The computation of the average loss interval using the discount factors is a simple modification of the procedure in Section 5.4, as follows:

```
I_tot0 = I_0 * w_0
I_tot1 = 0;
W_tot0 = w_0
W_tot1 = 0;
for (i = 1 to n-1) {
  I_tot0 = I_tot0 + (I_i * w_i * DF_i * DF);
  W_tot0 = W_tot0 + w_i * DF_i * DF;
}
for (i = 1 to n) {
  I_tot1 = I_tot1 + (I_i * w_(i-1) * DF_i);
  W_tot1 = W_tot1 + w_(i-1) * DF_i;
}
p = min(W_tot0/I_tot0, W_tot1/I_tot1);
```

The general discounting factor, `DF` is updated on every packet arrival as follows. First, a receiver computes the weighted average `I_mean` of the loss intervals `I_1, ..., I_n`:

```
I_tot = 0;
W_tot = 0;
for (i = 1 to n) {
  W_tot = w_(i-1) * DF_i;
  I_tot = I_tot + (I_i * w_(i-1) * DF_i);
}
I_mean = I_tot / W_tot;
```

This weighted average `I_mean` is compared to `I_0`, the number of packets received since the last loss event. If `I_0` is greater than twice `I_mean`, then the new loss interval is considerably larger than the old ones, and the general discount factor DF is updated to decrease the relative weight on the older intervals, as follows:

```
if (I_0 > 2 * I_mean) {
  DF = 2 * I_mean/I_0;
  if (DF < THRESHOLD)
    DF = THRESHOLD;
} else
  DF = 1;
```

A nonzero value for THRESHOLD ensures that older loss intervals from an earlier time of high congestion are not discounted entirely. We recommend a THRESHOLD of 0.5. Note that with each new packet arrival, I_0 will increase further, and the discount factor DF will be updated.

When a new loss event occurs, the current interval shifts from I_0 to I_1, loss interval I_i shifts to interval I_(i+1), and the loss interval I_n is forgotten. The previous discount factor DF has to be incorporated into the discount array. Because DF_i carries the discount factor associated with loss interval I_i, the DF_i array has to be shifted as well. This is done as follows:

```
for (i = 1 to n) {
  DF_i = DF * DF_i;
}
for (i = n-1 to 0 step -1) {
  DF_(i+1) = DF_i;
}
I_0 = 1;
DF_0 = 1;
DF = 1;
```

This completes the description of the optional history discounting mechanism. We emphasize that this is an optional mechanism whose sole purpose is to allow TFMCC to response somewhat more quickly to the sudden absence of congestion, as represented by a long current loss interval.

5.6. Initializing the Loss History after the First Loss Event

The number of packets received before the first loss event ususally does not reflect the current loss event rate. When the first loss event occurs, a TFMCC receiver assumes that the correct data rate is the rate at which data was received during the last RTT when the loss occurred. Instead of initializing the first loss interval to the number of packets sent until the first loss event, the TFMCC receiver calculates the loss interval that would be required to produce the receive rate X_recv, and uses this synthetic loss interval l_0 to seed the loss history mechanism.

The initial loss interval is calculated by inverting a simplified version of the TCP Equation (1).

$$X\_recv = \sqrt{3/2}\,\frac{s}{R\sqrt{1/l\_0}} \implies l\_0 = \left(\frac{X\_recv * R}{\sqrt{3/2}s}\right)^2$$

The resulting initial loss interval is too small at higher loss rates compared to using the more accurate Eqation (1), which leads to a more conservative initial loss event rate.

If a receiver still uses the initial RTT `R_max` instead of its real RTT, the initial loss interval is too large in case the initial RTT is higher than the actual RTT. As a consequence, the receiver will calculate a too high desired rate when the first RTT measurement `R` is made and the initial loss interval is still in the loss history. The receiver has to adjust `l_0` as follows:

$$l\_0 = l\_0 * (R/R\_max)^2$$

No action needs to be taken when the first RTT measurement is made after the initial loss interval left the loss history.

6. Security Considerations

TFMCC is not a transport protocol in its own right, but a congestion control mechanism that is intended to be used in conjunction with a transport protocol. Therefore security primarily needs to be considered in the context of a specific transport protocol and its authentication mechanisms.

Congestion control mechanisms can potentially be exploited to create denial of service. This may occur through spoofed feedback. Thus any transport protocol that uses TFMCC should take care to ensure that feedback is only accepted from valid receivers of the data. The precise mechanism to achieve this will however depend on the transport protocol itself.

Congestion control mechanisms may potentially be manipulated by a greedy receiver that wishes to receive more than its fair share of network bandwidth. However, in TFMCC a receiver can only influence the sending rate if it is the CLR and thus has the lowest calculated rate of all receivers. If the calculated rate is then manipulated such that it exceeds the calculated rate of the second to lowest receiver, it will cease to be CLR. A greedy receiver can only significantly increase then transmission rate if it is the only participant in the session. If such scenarios are of concern, possible defenses against such a receiver would normally include some form of nonce that the receiver must feed back to the sender to prove receipt. However, the details of such a nonce would depend on the transport protocol, and in particular on whether the transport protocol is reliable or unreliable.

It is possible that a receiver sends feedback claiming it has a very low calculated rate. This will reduce the rate of the multicast session and might render it useless but obviously cannot hurt the network itself.

We expect that protocols incorporating ECN with TFMCC will also want to incorporate feedback from the receiver to the sender using the ECN nonce [WES01]. The ECN nonce is a modification to ECN that protects the sender from the accidental or malicious concealment of marked packets. Again, the details of such a nonce would depend on the transport protocol, and are not addressed in this document.

7. IANA Considerations

There are no IANA actions required for this document.

8. Authors' Addresses

Joerg Widmer
Lehrstuhl Praktische Informatik IV
University of Mannheim
L 15, 16 - Room 415
D-68131 Mannheim
Germany
widmer@informatik.uni-mannheim.de

Mark Handley
ICSI Center for Internet Research
1947 Center St, Suite 600
Berkeley, CA 94708
mjh@icir.org

9. Acknowledgments

We would like to acknowledge feedback and discussions on equation-based congestion control with a wide range of people, including members of the Reliable Multicast Research Group, the Reliable Multicast Transport Working Group, and the End-to-End Research Group.

10. References

[1] B. Adamson, C. Bormann, M. Handley, and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Protocol Building Blocks", Internet Draft draft-ietf-rmt-norm-bb-02.txt, July 2001, work in progress. Citation for informational purposes only.

[2] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications", Proc ACM SIGCOMM 2000, Stockholm, August 2000

[3] H. W. Holbrook, "A Channel Model for Multicast," Ph.D. Dissertation, Stanford University, Department of Computer Science, Stanford, California, August 2001.

[4] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", Proc ACM SIGCOMM 1998.

[5] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[6] K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC 2481, January 1999.

[7] L. Rizzo, "pgmcc: a TCP-friendly single-rate multicast congestion control scheme", Proc ACM SIGCOMM 2000, Stockholm, August 2000

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[9] D. Wetherall, D. Ely, and N. Spring, "Robust ECN Signaling with Nonces", Internet Draft draft-ietf-tsvwg-tcp-nonce-00.txt, January 2001, work in progress. Citation for informational purposes only.

[10] J. Widmer and T. Fuhrmann, "Extremum Feedback for Very Large Multicast Groups", Proc NGC 2001, London, November 2001.

[11] J. Widmer and M. Handley, "Extending Equation-Based Congestion Control to Multicast Applications", Proc ACM SIGCOMM 2001, San Diego, August 2001

11. Full Copyright Statement