TSV WG
Mark Handley/ICIR
Jitendra Padhye/Microsoft
Sally Floyd/ICIR
Jörg Widmer/Univ. Mannheim

22 October 2002
Expires: April 2003

TCP Friendly Rate Control (TFRC): Protocol Specification

Status of this Document

This document is a product of the IETF TSV WG. Comments should be addressed to the authors.

Abstract

This document specifies TCP-Friendly Rate Control (TFRC). TFRC is a congestion control mechanism for unicast flows operating in a best-effort Internet environment. It is reasonably fair when competing for bandwidth with TCP flows, but has a much lower variation of throughput over time compared with TCP, making it more suitable for applications such as telephony or streaming media where a relatively smooth sending rate is of importance.

Table of Contents

NOTE TO RFC EDITOR: PLEASE DELETE THIS NOTE UPON PUBLICATION.
Changes from draft-ietf-tsvwg-tfrc-04.txt:
* In Section 4.3, changed a variable name from RTT to R, for consistency.

* In Section 4.4, made one change for clarity: Changed:
"Cut the sending rate in half. This is done by..."
to:
"Cut the sending rate in half. If the sender has received feedback from the receiver, this is done by..."

* In Section 4.4, made one change for correctness:
Changed:
"when the sender ... has not yet received any feedback from the sender,"
to:
"when the sender ... has not yet received any feedback from the receiver,"
Thanks to Mike Luby for these reports.

* Added a section about implementation-related issues.

* Made the following changes in response to feedback from the IESG:

* Changed the 64-seconds term in equations to "t_mbi", which is explained as representing 64 seconds. This is to make it clear that this term is in units of seconds.

* Added a section about a sender-based variant of TFRC.

Changes from draft-ietf-tsvwg-tfrc-03.txt:
* Changed one line in the computation of the weighted average I_mean of the loss intervals from:
W_tot = w_(i-1) * DF_i;
to:
W_tot = W_tot + w_(i-1) * DF_i;
Bug report from Wendy Lee.

* Removed the specification that the sender sends its current transmit rate. This was only used to initialize X_recv, and is not needed.

* Said that instead of sending the timestamp and the estimate of the RTT, the sender could send a coarse-grained counter that increments every quarter RTT. In this case, the receiver also does not have to send the timestamp back to the sender.
END OF NOTE TO RFC EDITOR.

1. Introduction

This document specifies TCP-Friendly Rate Control (TFRC). TFRC is a congestion control mechanism designed for unicast flows operating in a Internet environment and competing with TCP traffic [2]. 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 [7], in an application incorporating end-to-end congestion control at the application level, or in the context of endpoint congestion management [1]. This document does not discuss packet formats or reliability. Implementation-related issues are discussed only briefly, in Section 8.

TFRC is designed to be reasonably fair when competing for bandwidth with TCP flows, where a flow is ‘‘reasonably fair’’ if its sending rate is generally within a factor of two of the sending rate of a TCP flow under the same conditions. However TFRC has a much lower variation of throughput over time compared with TCP, which makes it more suitable for applications such as telephony or streaming media where a relatively smooth sending rate is of importance.

The penalty of having smoother throughput than TCP while competing fairly for bandwidth is that TFRC responds slower than TCP to changes in available bandwidth. Thus TFRC should only be used when the application has a requirement for smooth throughput, in particular, avoiding

TCP's halving of the sending rate in response to a single packet drop. For applications that simply need to transfer as much data as possible in as short a time as possible we recommend using TCP, or if reliability is not required, using an Additive-Increase, Multiplicative-Decrease (AIMD) congestion control scheme with similar parameters to those used by TCP.

TFRC 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; TFRC-PS (for TFRC-PacketSize) is a variant of TFRC for applications that have a fixed sending rate but vary their packet size in response to congestion. TFRC-PS will be specified in a later document.

TFRC is a receiver-based mechanism, with the calculation of the congestion control information (i.e., the loss event rate) in the data receiver rather in the data sender. This is well-suited to an application where the sender is a large server handling many concurrent connections, and the receiver has more memory and CPU cycles available for computation. In addition, a receiver-based mechanism is more suitable as a building block for multicast congestion control.

2. 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 TFRC implementations.

3. Protocol Mechanism

For its congestion control mechanism, TFRC directly uses a throughput equation for the allowed sending rate as a function of the loss event rate and round-trip time. In order to compete fairly with TCP, TFRC uses the TCP throughput equation, which roughly describes TCP's sending rate as a function of the loss event rate, round-trip time, and packet size. We define a loss event as one or more lost or marked packets from a window of data, where a marked packet refers to a congestion indication from Explicit Congestion Notification (ECN) [6].

Generally speaking, TFRC's congestion control mechanism works as follows:

- The receiver measures the loss event rate and feeds this information back to the sender.
- The sender also uses these feedback messages to measure the round-trip time (RTT).
- The loss event rate and RTT are then fed into TFRC's throughput equation, giving the acceptable transmit rate.
- The sender then adjusts its transmit rate to match the calculated rate.

The dynamics of TFRC are sensitive to how the measurements are performed and applied. 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 TFRC.

3.1. TCP Throughput Equation

Any realistic equation giving TCP throughput as a function of loss event rate and RTT should be suitable for use in TFRC. 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 TFRC is a slightly simplified version of the throughput equation for Reno TCP from [4]. Ideally we'd prefer a throughput equation based on SACK TCP, but no one has yet derived the throughput equation for SACK TCP, and from both simulations and experiments, the differences between the two equations are relatively minor.

The throughput equation is:

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

Where:

X is the transmit rate in bytes/second.

s is the packet size in bytes.

R is the round trip time in seconds.

p is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted.

t_RTO is the TCP retransmission timeout value in seconds.

b is the number of packets acknowledged by a single TCP acknowledgement.

We further simplify this by setting `t_RTO = 4*R`. A more accurate calculation of *t_RTO* is possible, but experiments with the current setting have resulted in reasonable fairness with existing TCP implementations [9]. Another possibility would be to set `t_RTO` = max(4`R, one second)`, to match the recommended minimum of one second on the RTO [5].

Many current TCP connections use delayed acknowledgements, sending an acknowledgement for every two data packets received, and thus have a sending rate modeled by $b = 2$. However, TCP is also allowed to send an acknowledgement for every data packet, and this would be modeled by $b = 1$. Because many TCP implementations do not use delayed acknowledgements, we recommend $b = 1$.

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 TFRC implementation. The measurement of s is specified in Section 4.1, measurement of R is specified in Section 4.3, and measurement of p is specified in Section 5. In the rest of this document all data rates are measured in bytes/second.

3.2. Packet Contents

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

3.2.1. Data Packets

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

- A sequence number. 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.
- A timestamp indicating when the packet is sent. We denote by `ts_i` the timestamp of the packet with sequence number `i`. The resolution of the timestamp should typically be measured in milliseconds.
  This timestamp is used by the receiver to determine which losses belong to the same loss event. The timestamp is also echoed by the receiver to enable the sender to estimate the round-trip time, for senders that do not save timestamps of transmitted data packets.
  We note that as an alternative to a timestamp incremented in milliseconds, a "timestamp" that increments every quarter of a round-trip time would be sufficient for determining when losses belong to the same loss event, in the context of a protocol where this is understood by both sender and receiver, and where the sender saves the timestamps of transmitted data packets.
- The sender's current estimate of the round trip time. The estimate reported in packet `i` is denoted by `R_i`. The round-trip time estimate is used by the receiver, along with the timestamp, to determine when multiple losses belong to the same loss event.
  If the sender sends a coarse-grained "timestamp" that increments every quarter of a round-trip time, as discussed above, then the sender does not need to send its current estimate of the round trip time.

3.2.2. Feedback Packets

Each feedback packet sent by the data receiver contains the following information:

- The timestamp of the last data packet received. We denote this by `t_recvdata`. If the last packet received at the receiver has sequence number `i`, then `t_recvdata = ts_i`.

This timestamp is used by the sender to estimate the round-trip time, and is only needed if the sender does not save timestamps of transmitted data packets.

- The amount of time elapsed between the receipt of the last data packet at the receiver, and the generation of this feedback report. We denote this by `t_delay`.
- The rate at which the receiver estimates that data was received since the last feedback report was sent. We denote this by `X_recv`.
- The receiver's current estimate of the loss event rate, `p`.

4. Data Sender Protocol

The data sender sends a stream of data packets to the data receiver at a controlled rate. When a feedback packet is received from the data receiver, the data sender changes its sending rate, based on the information contained in the feedback report. If the sender does not receive a feedback report for two round trip times, it cuts its sending rate in half. This is achieved by means of a timer called the *nofeedback* timer.

We specify the sender-side protocol in the following steps:

- Measurement of the mean packet size being sent.
- The sender behavior when a feedback packet is received.
- The sender behavior when the *nofeedback* timer expires.
- Oscillation prevention (optional)
- Scheduling of transmission on non-realtime operating systems.

4.1. Measuring the Packet Size

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 completely different way of measuring parameters.

The second class of applications are discussed separately in a separate document on TFRC-PS. For the remainder of this section we assume the sender can estimate the packet size, and that congestion control is performed by adjusting the number of packets sent per second.

4.2. Sender Initialization

To initialize the sender, the value of X is set to 1 packet/second and the nofeedback timer is set to expire after 2 seconds. The initial values for R (RTT) and t_RTO are undefined until they are set as described below. The intial value of tld, for the Time Last Doubled during slow-start, is set to -1.

4.3. Sender behavior when a feedback packet is received

The sender knows its current sending rate, X, and maintains an estimate of the current round trip time, R, and an estimate of the timeout interval, t_RTO.

When a feedback packet is received by the sender at time t_now, the following actions should be performed:

1) Calculate a new round trip sample.
   R_sample = (t_now - t_recvdata) - t_delay.

2) Update the round trip time estimate:

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

   TFRC is not sensitive to the precise value for the filter constant q, but we recommend a default value of 0.9.

3) Update the timeout interval:

```
t_RTO = 4*R.
```

4) Update the sending rate as follows:

```
If (p > 0)
    Calculate X_calc using the TCP throughput equation.
    X = max(min(X_calc, 2*X_recv), s/t_mbi);
Else
    If (t_now - tld >= R)
        X = max(min(2*X, 2*X_recv), s/R);
        tld = t_now;
```

   Note that if p == 0, then the sender is in slow-start phase, where it approximately doubles the sending rate each round-trip time until a loss occurs. The s/R term gives a minimum sending rate during slow-start of one packet per RTT. The parameter t_mbi is 64 seconds, and represents the maximum inter-packet backoff interval in the persistent absense of feedback. Thus, when p > 0 the sender sends at least one packet every 64 seconds.

5) Reset the `nofeedback` timer to expire after `max(4*R, 2*s/X)` seconds.

4.4. Expiration of nofeedback timer

If the `nofeedback` timer expires, the sender should perform the following actions:

1) Cut the sending rate in half. If the sender has received feedback from the receiver, this is done by modifying the sender's cached copy of X_recv (the receive rate). Because the sending rate is limited to at most twice X_recv, modifying X_recv limits the current sending rate, but allows the sender to slow-start, doubling its sending rate each RTT, if feedback messages resume reporting no losses.

```
If (X_calc > 2*X_recv)
    X_recv = max(X_recv/2, s/(2*t_mbi));
Else
    X_recv = X_calc/4;
```

The term `s/(2*t_mbi)` limits the backoff to one packet every 64 seconds in the case of persistent absence of feedback.

2) The value of `X` must then be recalculated as described under point (4) above.

If the `nofeedback` timer expires when the sender does not yet have an RTT sample, and has not yet received any feedback from the receiver, then step (1) can be skipped, and the sending rate cut in half directly:

$$X = \max(X/2, s/t\_mbi)$$

3) Restart the `nofeedback` timer to expire after `max(4*R, 2*s/X)` seconds.

Note that when the sender stops sending, the receiver will stop sending feedback. This will cause the `nofeedback` timer to start to expire and decrease `X_recv`. If the sender subsequently starts to send again, `X_recv` will limit the transmit rate, and a normal slowstart phase will occur until the transmit rate reaches `X_calc`.

4.5. Preventing Oscillations

To prevent oscillatory behavior in environments with a low degree of statistical multiplexing it is useful to modify sender's transmit rate to provide congestion avoidance behavior by reducing the transmit rate as the queuing delay (and hence RTT) increases. To do this the sender maintains an estimate of the long-term RTT and modifies its sending rate depending on how the most recent sample of the RTT differs from this value. The long-term sample is `R_sqmean`, and is set as follows:

```
If no feedback has been received before
    R_sqmean = sqrt(R_sample);
Else
    R_sqmean = q2*R_sqmean + (1-q2)*sqrt(R_sample);
```

Thus `R_sqmean` gives the exponentially weighted moving average of the square root of the RTT samples. The constant `q2` should be set similarly to `q`, and we recommend a value of 0.9 as the default.

The sender obtains the base transmit rate, `X`, from the throughput function. It then calculates a modified instantaneous transmit rate `X_inst`, as follows:

```
X_inst = X * R_sqmean / sqrt(R_sample);
```

When `sqrt(R_sample)` is greater than `R_sqmean` then the queue is typically increasing and so the transmit rate needs to be decreased for stable operation.

Note: This modification is not always strictly required, especially if the degree of statistical multiplexing in the network is high. However we recommend that it is done because it does make TFRC behave better in environments with a low level of statistical multiplexing. If it is not done, we recommend using a very low value of `q`, such that `q` is close to or exactly zero.

4.6. Scheduling of Packet Transmissions

As TFRC 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 course-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_inst;
```

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.

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

5. Calculation of the Loss Event Rate (p)

Obtaining an accurate and stable measurement of the loss event rate is of primary importance for TFRC. Loss rate measurement is performed at the receiver, based on the detection of lost or marked packets from the sequence numbers of arriving packets. We describe this process before describing the rest of the receiver protocol.

5.1. Detection of Lost or Marked Packets

TFRC 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 receiver maintains 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 receiver 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 TFRC more robust in the presence of reordering. In contrast to TCP, if a packet arrives late (after 3 subsequent packets arrived) in TFRC, the late packet can fill the hole in TFRC's reception record, and the receiver can recalculate the loss event rate. Future versions of TFRC 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

TFRC requires that the loss fraction 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 receiver needs to map the packet loss history into a loss event record, where a loss event is one or more packets lost in an RTT. To perform this mapping, the receiver needs to know the RTT to use, and this is supplied periodically by the sender, typically as control information piggy-backed onto a data packet.

TFRC is not sensitive to how the RTT measurement sent to the receiver is made, but we recommend using the sender's calculated RTT, R, (see Section 4.3) for this purpose.

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 this 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 in 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 TRFC's speed in responding to changes in the level of congestion. As currently specified, TFRC 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 TFRC'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 `1/(0.75*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, discussed further in [3] and [9], that allows the TFRC receiver 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, for i > 0`, 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, the 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_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 TFRC to response somewhat more quickly to the sudden absence of congestion, as represented by a long current loss interval.

6. Data Receiver Protocol

The receiver periodically sends feedback messages to the sender. Feedback packets should normally be sent at least once per RTT, unless the sender is sending at a rate of less than one packet per RTT, in which case a feedback packet should be send for every data packet received. A feedback packet should also be sent whenever a new loss event is detected without waiting for the end of an RTT, and whenever an out-of-order data packet is received that removes a loss event from the history.

If the sender is transmitting at a high rate (many packets per RTT) there may be some advantages to sending periodic feedback messages more than once per RTT as this allows faster response to changing RTT measurements, and more resilience to feedback packet loss. However there is little gain from sending a large number of feedback messages per RTT.

6.1. Receiver behavior when a data packet is received

When a data packet is received, the receiver performs the following steps:

1) Add the packet to the packet history.

2) Let the previous value of `p` be `p_prev`. Calculate the new value of `p` as described in Section 5.

3) If `p > p_prev`, cause the feedback timer to expire, and perform the actions described in Section 6.2

   If `p <= p_prev` no action need be performed.

   However an optimization might check to see if the arrival of the packet caused a hole in the packet history to be filled and consequently two loss intervals were merged into one. If this is the case, the receiver might also send feedback immediately. The effects of such an optimization are normally expected to be small.

6.2. Expiration of feedback timer

When the feedback timer at the receiver expires, the action to be taken depends on whether data packets have been received since the last feedback was sent.

Let the maximum sequence number of a packet at the receiver so far be `S_m`, and the value of the RTT measurement included in packet `S_m` be R_m. If data packets have been received since the pervious feedback was sent, the receiver performs the following steps:

1) Calculate the average loss event rate using the algorithm described above.

2) Calculate the measured receive rate, `X_recv`, based on the packets received within the previous `R_m` seconds.

3) Prepare and send a feedback packet containing the information described in Section 3.2.2

4) Restart the feedback timer to expire after `R_m` seconds.

If no data packets have been received since the last feedback was sent, no feedback packet is sent, and the feedback timer is restarted to expire after R_m seconds.

6.3. Receiver initialization

The receiver is initialized by the first packet that arrives at the receiver. Let the sequence number of this packet be i.

When the first packet is received:

- Set p=0
- Set X_recv = 0.
- Prepare and send a feedback packet.
- Set the feedback timer to expire after R_i seconds.

6.3.1. Initializing the Loss History after the First Loss Event

The number of packets until the first loss can not be used to compute the sending rate directly, as the sending rate changes rapidly during this time. TFRC assumes that the correct data rate after the first loss is half of the sending rate when the loss occurred. TFRC approximates this target rate by X_recv, the receive rate over the most recent round-trip time. After the first loss, instead of initializing the first loss interval to the number of packets sent until the first loss, the TFRC receiver calculates the loss interval that would be required to produce the data rate X_recv, and uses this synthetic loss interval to seed the loss history mechanism.

TFRC does this by finding some value p for which the throughput equation in Section 3.1 gives a sending rate within 5% of X_recv, given the current packet size s and round-trip time R. The first loss interval is then set to 1/p. (The 5% tolerance is introduced simply because the throughput equation is difficult to invert, and we want to reduce the costs of calculating p numerically.)

7. Sender-based Variants

It would be possible to implement a sender-based variant of TFRC, where the receiver uses reliable delivery to send information about packet losses to the sender, and the sender computes the packet loss rate and the acceptable transmit rate. However, we do not specify the details of a sender-based variant in this document.

The main advantages of a sender-based variant of TFRC would be that the sender would not have to trust the receiver’s calculation of the packet loss rate. However, with the requirement of reliable delivery of loss information from the receiver to the sender, a sender-based TFRC would have much tighter constraints on the transport protocol in which it is embedded.

In contrast, the receiver-based variant of TFRC specified in this document is robust to the loss of feedback packets, and therefore does not require the reliable delivery of feedback packets. It is also better suited for applications such as streaming media from web servers, where it is typically desirable to offload work from the server to the client as much as possible.

The sender-based and receiver-based variants also have different properties in terms of upgrades. For example, for changes in the procedure for calculating the packet loss rate, the sender would have to be upgraded in the sender-based variant, and the receiver would have to be upgraded in the receiver-based variant.

8. Implementation Issues

This document has specified the TFRC congestion control mechanism, for use by applications and transport protocols. This section mentions briefly some of the few implementation issues.

For `t_RTO = 4*R` and $b = 1$, the throughput equation in Section 3.1 can be expressed as follows:

$$X = \frac{s}{R * f(p)}$$

for

$$f(p) = \sqrt{\frac{2p}{3}} + (12\sqrt{\frac{3p}{8}})p(1 + 32p^2)$$

A table lookup could be used for the function f(p).

Many of the multiplications (e.g., q and $1-q$ for the round-trip time average, a factor of 4 for the timeout interval) are or could be by powers of two, and therefore could be implemented as simple shift operations.

We note that the optional sender mechanism for preventing oscillations described in Section 4.5 uses a square-root computation.

The calculation of the average loss interval in Section 5.4 involves multiplications by the weights `w_0` to `w_(n-1)`, which for n=8 are:

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

With a minor loss of smoothness, it would be possible to use weights that were powers of two or sums of powers of two, e.g.,

1.0, 1.0, 1.0, 1.0, 0.75, 0.5, 0.25, 0.25.

The optional history discounting mechanism described in Section 5.5 is used in the calculation of the average loss rate. The history discounting mechanism is invoked only when there has been an unusually long interval with no packet losses. For a more efficient operation, the discount factor `DF_i` could be restricted to be a power of two.

9. Security Considerations

TFRC 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 TFRC should take care to ensure that feedback is only accepted from the receiver of the data. The precise mechanism to achieve this will however depend on the transport protocol itself.

In addition, congestion control mechanisms may potentially be manipulated by a greedy receiver that wishes to receive more than its fair share of network bandwidth. A receiver might do this by claiming to have received packets that in fact were lost due to congestion. 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.

We expect that protocols incorporating ECN with TFRC will also want to incorporate feedback from the receiver to the sender using the ECN nonce [WES02]. 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.

10. IANA Considerations

There are no IANA actions required for this document.

11. Authors' Addresses

Mark Handley, Sally Floyd
ICIR/ICSI
1947 Center St, Suite 600
Berkeley, CA 94708
mjh@icir.org, floyd@icir.org

Jitendra Padhye
Microsoft Research
padhye@microsoft.com

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

12. 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. We would like to thank Ken Lofgren, Mike Luby, Eduardo Urzaiz, Vladica Stanisic, Randall Stewart, Shushan Wen, and Wendy Lee (lhh@zsu.edu.cn) for feedback on earlier versions of this document, and to thank Mark Allman for his extensive feedback from using the draft to produce a working implementation.

13. Normative References

14. Non-normative References

[1] Balakrishnan, H., Rahul, H., and Seshan, S., "An Integrated Congestion Management Architecture for Internet Hosts," Proc. ACM SIGCOMM, Cambridge, MA, September 1999.

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

[3] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications: the Extended Version", ICSI tech report TR-00-03, March 2000.

[4] Padhye, J. and Firoiu, V. and Towsley, D. and Kurose, J., "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, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

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

[8] Wetherall, D., Ely, D., and Spring, N., "Robust ECN Signaling with Nonces", draft-ietf-tsvwg-tcp-nonce-03.txt, internet draft, work in progress, April 2002.

[9] Widmer, J., "Equation-Based Congestion Control", Diploma Thesis, University of Mannheim, February 2000. URL "http://www.icir.org/tfrc/".