Network Working Group
Request for Comments: 5348
Obsoletes: 3448
Updates: 4342
S. Floyd
ICIR
M. Handley
University College London
J. Padhye
Microsoft
J. Widmer
DoCoMo
September 2008

TCP Friendly Rate Control (TFRC): Protocol Specification

Status of This Memo

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 streaming media where a relatively smooth sending rate is of importance.

This document obsoletes RFC 3448 and updates RFC 4342.

Table of Contents

1. Introduction

This document specifies TCP Friendly Rate Control (TFRC). TFRC is a congestion control mechanism designed for unicast flows operating in an Internet environment and competing with TCP traffic [FHPW00]. Instead of specifying a complete protocol, this document simply specifies a congestion control mechanism that could be used in a transport protocol such as DCCP (Datagram Congestion Control Protocol) [RFC4340], in an application incorporating end-to-end congestion control at the application level, or in the context of endpoint congestion management [BRS99]. 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 we call a flow "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 best performance with applications that use a fixed segment size, and vary their sending rate in packets per second in response to congestion. TFRC can also be used, perhaps with less optimal performance, with applications that do not have a fixed segment size, but where the segment size varies according to the needs of the application (e.g., video applications).

Some applications (e.g., some audio applications) require a fixed interval of time between packets and vary their segment size instead of their packet rate in response to congestion. The congestion control mechanism in this document is not designed for those applications; TFRC-SP (Small-Packet TFRC) is a variant of TFRC for applications that have a fixed sending rate in packets per second but either use small packets or vary their packet size in response to congestion. TFRC-SP is specified in a separate document [RFC4828].

This document specifies TFRC as 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. However, it is also possible to implement TFRC in sender-based variants, as allowed in DCCP's Congestion Control ID 3 (CCID 3) [RFC4342].

This document obsoletes RFC 3448. In the transport protocol DCCP (Datagram Congestion Control Protocol) [RFC4340], the Congestion Control ID Profiles CCID-3 [RFC4342] and CCID-4 [CCID-4] both specify the use of TFRC from RFC 3448. CCID-3 and CCID-4 implementations SHOULD use this document instead of RFC 3448 for the specification of TFRC.

The normative specification of TFRC is in Sections 3-6. Section 7 discusses sender-based variants, Section 8 discusses implementation issues, and Section 9 gives a non-normative overview of differences with RFC 3448.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Appendix A gives a list of technical terms used in this document.

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 segment 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) [RFC3168].

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, and the resulting sending rate is limited to at most twice the receive rate to give the allowed transmit rate X.

- The sender then adjusts its transmit rate to match the allowed transmit rate X.

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 currently REQUIRED for TFRC is a slightly simplified version of the throughput equation for Reno TCP from [PFTK98]. Ideally, we would prefer a throughput equation based on selective acknowledgment (SACK) TCP, but no one has yet derived the throughput equation for SACK TCP, and simulations and experiments suggest that the differences between the two equations would be relatively minor [FF99] (Appendix B).

The throughput equation for X_Bps, TCP's average sending rate in bytes per second, is:

```
                              s
X_Bps = ----------------------------------------------------------
        R*sqrt(2*b*p/3) + (t_RTO * (3*sqrt(3*b*p/8)*p*(1+32*p^2)))
```

Where:

X_Bps is TCP's average transmit rate in bytes per second. (X_Bps is the same as X_calc in RFC 3448.)

s is the segment size in bytes (excluding IP and transport protocol headers).

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 maximum number of packets acknowledged by a single TCP acknowledgement.

Setting the TCP retransmission timeout value t_RTO:
Implementations SHOULD set t_RTO = 4*R. Implementations MAY choose to implement a more accurate calculation of t_RTO. Implementations MAY also set t_RTO to max(4*R, one second), to match the recommended minimum of one second on the RTO [RFC2988].

Setting the parameter b for delayed acknowledgements:
Some current TCP connections use delayed acknowledgements, sending an acknowledgement for every two data packets received. However, TCP is also allowed to send an acknowledgement for every data packet. For the revised TCP congestion control mechanisms, [RFC2581bis] currently specifies that the delayed acknowledgement algorithm should be used with TCP. However, [RFC2581bis] recommends increasing the congestion window during congestion avoidance by one segment per RTT even in the face of delayed acknowledgements, consistent with a TCP throughput equation with b = 1. On an experimental basis, [RFC2581bis] allows for increases of the congestion window during slow-start that are also consistent with a TCP throughput equation with b = 1. Thus, the use of b = 1 is consistent with [RFC2581bis]. The use of b = 1 is RECOMMENDED.

With t_RTO=4*R and b=1, the throughput equation for X_Bps, the TCP sending rate in bytes per second, can be simplified as:

```
                                s
X_Bps = -----------------------------------------------
        R * (sqrt(2*p/3) + 12*sqrt(3*p/8)*p*(1+32*p^2))
```

In the future, updates to this document could specify different TCP equations to 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 throughput equation can also be expressed in terms of X_pps, the sending rate in packets per second, with

$$X\_pps = X\_Bps / s .$$

The parameters s (segment 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, the measurement of R is specified in Section 4.3, and the measurement of p is specified in Section 5. In the rest of this document, data rates are measured in bytes per second unless otherwise specified.

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 MUST be 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 MAY be used 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. The round-trip time estimate is also used by the receiver to determine the interval to use for calculating the receive rate and to determine when to send feedback packets.

  If the sender sends a coarse-grained "timestamp" that increments every quarter of a round-trip time, as discussed above, then the sender is not required 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 the 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 in the previous round-trip time. 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 four round-trip times, then the sender 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:

- o Measurement of the mean segment size being sent.
- o Sender initialization.
- o The sender behavior when a feedback packet is received.
- o The sender behavior when the nofeedback timer expires.
- o Oscillation prevention (optional).
- o Scheduling of packet transmission and allowed burstiness.

4.1. Measuring the Segment Size

The TFRC sender uses the segment size, s, in the throughput equation, in the setting of the maximum receive rate, the setting of the minimum and initial sending rates, and the setting of the nofeedback timer. The TFRC receiver MAY use the average segment size, s, in initializing the loss history after the first loss event. As specified in Section 6.3.1, if the TFRC receiver does not know the segment size, s, used by the sender, the TFRC receiver MAY instead use the arrival rate in packets per second in initializing the loss history.

The segment size is normally known to an application. This may not be so in two cases:

1) The segment size naturally varies depending on the data. In this case, although the segment size varies, that variation is not coupled to the transmit rate. The TFRC sender can either compute the average segment size or use the maximum segment size for the segment size, s.

2) The application needs to change the segment size rather than the number of segments 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.

For the first class of applications where the segment size varies depending on the data, the sender SHOULD estimate the segment size, s, as the average segment size over the last four loss intervals. The sender MAY estimate the average segment size over longer time intervals, if so desired.

The second class of applications are discussed separately in a separate document on TFRC-SP [RFC4828]. For the remainder of this section we assume the sender can estimate the segment size and that congestion control is performed by adjusting the number of packets sent per second.

4.2. Sender Initialization

The initial values for X (the allowed sending rate in bytes per second) and tld (the Time Last Doubled during slow-start, in seconds) are undefined until they are set as described below. If the sender is ready to send data when it does not yet have a round-trip sample, the value of X is set to s bytes per second, for segment size s, the nofeedback timer is set to expire after two seconds, and tld is set to 0 (or to -1, either one is okay). Upon receiving the first round-trip time measurement (e.g., after the first feedback packet or the SYN exchange from the connection setup, or from a previous connection [RFC2140]), tld is set to the current time, and the allowed transmit rate, X, is set to the initial_rate, specified as W_init/R, for W_init based on [RFC3390]:

```
initial_rate = W_init/R; W_init = min(4*MSS, max(2*MSS, 4380)).
```

In computing W_init, instead of using Maximum Segment Size (MSS), the TFRC sender SHOULD use the maximum segment size to be used for the initial round-trip time of data, if that is known by the TFRC sender when X is initialized.

For responding to the initial feedback packet, this replaces step (4) of Section 4.3 below.

Appendix B explains why the initial value of TFRC's nofeedback timer is set to two seconds, instead of the recommended initial value of three seconds for TCP's retransmit timer from [RFC2988].

4.3. Sender Behavior When a Feedback Packet Is Received

The sender knows its current allowed sending rate, X, and maintains an estimate of the current round-trip time R. The sender also maintains X_recv_set as a small set of recent X_recv values (typically only two values).

Initialization: X_recv_set is first initialized to contain a single item, with value Infinity. (As an implementation-specific issue, X_recv_set MAY be initialized to a large number instead of to Infinity, e.g., to the largest integer that is easily representable.)

When a feedback packet is received by the sender at time t_now, the current time in seconds, the following actions MUST be performed.

1) Calculate a new round-trip sample:

```
R_sample = (t_now - t_recvdata) - t_delay.
```

As described in Section 3.2.2, t_delay gives the elapsed time at the receiver.

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 a default value of 0.9 is RECOMMENDED.

3) Update the timeout interval:

```
RTO = max(4*R, 2*s/X)
```

4) Update the allowed sending rate as follows. This procedure uses the variables t_mbi and recv_limit:

```
t_mbi: the maximum backoff interval of 64 seconds.
recv_limit: the limit on the sending rate computed from
                  X_recv_set.
```

This procedure also uses the procedures Maximize X_recv_set() and Update X_recv_set(), which are defined below.

The procedure for updating the allowed sending rate:

```
If (the entire interval covered by the feedback packet
      was a data-limited interval) {
    If (the feedback packet reports a new loss event or an
                 increase in the loss event rate p) {
        Halve entries in X_recv_set;
        X_recv = 0.85 * X_recv;
        Maximize X_recv_set();
        recv_limit = max (X_recv_set);
    } Else {
        Maximize X_recv_set();
        recv_limit = 2 * max (X_recv_set);
    }
} Else {                          // typical behavior
    Update X_recv_set();
    recv_limit = 2 * max (X_recv_set);
}
If (p > 0) {          // congestion avoidance phase
    Calculate X_Bps using the TCP throughput equation.
    X = max(min(X_Bps, recv_limit), s/t_mbi);
} Else if (t_now - tld >= R) {
    // initial slow-start
    X = max(min(2*X, recv_limit), initial_rate);
    tld = t_now;
}
```

5) If oscillation reduction is used, calculate the instantaneous transmit rate, X_inst, following Section 4.5.

6) Reset the nofeedback timer to expire after RTO seconds.

The procedure for maximizing X_recv_set keeps a single value, the largest value from X_recv_set and the new X_recv.

```
Maximize X_recv_set():
    Add X_recv to X_recv_set;
    Delete initial value Infinity from X_recv_set,
       if it is still a member.
    Set the timestamp of the largest item to the current time;
    Delete all other items.
```

The procedure for updating X_recv_set keeps a set of X_recv values with timestamps from the two most recent round-trip times.

```
Update X_recv_set():
    Add X_recv to X_recv_set;
    Delete from X_recv_set values older than
        two round-trip times.
```

Definition of a data-limited interval:
We define a sender as data-limited any time it is not sending as much as it is allowed to send. We define an interval as a 'data-limited interval' if the sender was data-limited over the *entire* interval; Section 8.2.1 discusses implementation issues for a sender in determining if an interval was a data-limited interval. The term 'data-limited interval' is used in the first "if" condition in step (4), which prevents a sender from having to reduce its sending rate as a result of a feedback packet reporting the receive rate from a data-limited period.

As an example, consider a sender that is sending at its full allowed rate, except that it is sending packets in pairs, rather than sending each packet as soon as it can. Such a sender is considered data-limited part of the time, because it is not always sending packets as soon as it can. However, consider an interval that covers this sender's transmission of at least two data packets; such an interval does not meet the definition of a data-limited interval because the sender was not data-limited *over the entire interval*.

If the feedback packet reports a receive rate X_recv of zero (i.e., the first feedback packet), the sender does not consider that the entire interval covered by the feedback packet was a data-limited interval.

X_recv_set and the first feedback packet:
Because X_recv_set is initialized with a single item, with value Infinity, recv_limit is set to Infinity for the first two round-trip times of the connection. As a result, the sending rate is not limited by the receive rate during that period. This avoids the problem of the sending rate being limited by the value of X_recv from the first feedback packet.

The interval covered by a feedback packet:
How does the sender determine the period covered by a feedback packet? This is discussed in more detail in Section 8.2. In general, the receiver will be sending a feedback packet once per round-trip time; so typically, the sender will be able to determine exactly the period covered by the current feedback packet from the previous feedback packet. However, in cases when the previous

feedback packet was lost, or when the receiver sends a feedback packet early because it detected a lost or ECN-marked packet, the sender will have to estimate the interval covered by the feedback packet. As specified in Section 6.2, each feedback packet sent by the receiver covers a round-trip time, for the round-trip time estimate R_m maintained by the receiver R_m seconds before the feedback packet was sent.

The response to a loss during a data-limited interval:
In TFRC, after the initial slow-start, the sender always updates the calculated transmit rate, X_Bps, after a feedback packet is received, and the allowed sending rate, X, is always limited by X_Bps. However, during a data-limited interval, when the actual sending rate is usually below X_Bps, the sending rate is still limited by recv_limit, derived from X_recv_set. If the sender is data-limited, possibly with a varying sending rate from one round-trip time to the next, and is experiencing losses, then we decrease the entry in X_recv_set in order to reduce the allowed sending rate.

The sender can detect a loss event during a data-limited period either from explicit feedback from the receiver, or from a reported increase in the loss event rate. When the sender receives a feedback packet reporting such a loss event in a data-limited interval, the sender limits the allowed increases in the sending rate during the data-limited interval.

The initial slow-start phase:
Note that when p=0, the sender has not yet learned of any loss events, and the sender is in the initial slow-start phase. In this initial slow-start phase, the sender can approximately double the sending rate each round-trip time until a loss occurs. The initial_rate term in step (4) gives a minimum allowed sending rate during slow-start of the initial allowed sending rate.

We note that if the sender is data-limited during slow-start, or if the connection is limited by the path bandwidth, then the sender is not necessarily able to double its sending rate each round-trip time; the sender's sending rate is limited to at most twice the past receive rate, or at most initial_rate, whichever is larger. This is similar to TCP's behavior, where the sending rate is limited by the rate of incoming acknowledgement packets as well as by the congestion window. Thus, in TCP's slow-start, for the most aggressive case of the TCP receiver acknowledging every data packet, the TCP sender's sending rate is limited to at most twice the rate of these incoming acknowledgment packets.

The minimum allowed sending rate:
The term s/t_mbi ensures that when p > 0, the sender is allowed to send at least one packet every 64 seconds.

4.4. Expiration of Nofeedback Timer

This section specifies the sender's response to a nofeedback timer. The nofeedback timer could expire because of an idle period or because of data or feedback packets dropped in the network.

This section uses the variable recover_rate. If the TFRC sender has been idle ever since the nofeedback timer was set, the allowed sending rate is not reduced below the recover_rate. For this document, the recover_rate is set to the initial_rate (specified in Section 4.2). Future updates to this specification may explore other possible values for the recover_rate.

If the nofeedback timer expires, the sender MUST perform the following actions:

1) Cut the allowed sending rate in half.

   If the nofeedback timer expires when the sender has had at least one RTT measurement, the allowed sending rate is reduced by modifying X_recv_set as described in the pseudocode below (including item (2)). In the general case, the sending rate is limited to at most twice X_recv. Modifying X_recv_set limits the sending rate, but still allows the sender to slow-start, doubling its sending rate each RTT, if feedback messages resume reporting no losses.

   If the sender has been idle since this nofeedback timer was set and X_recv is less than the recover_rate, then the allowed sending rate is not halved, and X_recv_set is not changed. This ensures that the allowed sending rate is not reduced to less than half the recover_rate as a result of an idle period.

   In the general case, the allowed sending rate is halved in response to the expiration of the nofeedback timer. The details, in the pseudocode below, depend on whether the sender is in slow-start, is in congestion avoidance limited by X_recv, or is in congestion avoidance limited by the throughput equation.

```
X_recv = max (X_recv_set);
If (sender does not have an RTT sample,
    has not received any feedback from receiver,
    and has not been idle ever since the nofeedback timer
    was set) {
    // We do not have X_Bps or recover_rate yet.
    // Halve the allowed sending rate.
    X = max(X/2, s/t_mbi);
} Else if (((p>0 && X_recv < recover_rate) or
      (p==0 && X < 2 * recover_rate)), and
       sender has been idle ever
       since nofeedback timer was set) {
    // Don't halve the allowed sending rate.
    Do nothing;
} Else if (p==0) {
    // We do not have X_Bps yet.
    // Halve the allowed sending rate.
    X = max(X/2, s/t_mbi);
} Else if (X_Bps > 2*X_recv)) {
    // 2*X_recv was already limiting the sending rate.
    // Halve the allowed sending rate.
    Update_Limits(X_recv;)
} Else {
    // The sending rate was limited by X_Bps, not by X_recv.
    // Halve the allowed sending rate.
    Update_Limits(X_Bps/2);
}
```

The term s/t_mbi limits the backoff to one packet every 64 seconds.

The procedure Update_Limits() uses the variable timer_limit for the limit on the sending rate computed from the expiration of the nofeedback timer, as follows:

```
Update_Limits(timer_limit):
    If (timer_limit < s/t_mbi)
        timer_limit = s/t_mbi;
    Replace X_recv_set contents with the single item
         timer_limit/2;
    Recalculate X as in step (4) of Section 4.3;
```

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

If the sender has been data-limited but not idle since the nofeedback timer was set, it is possible that the nofeedback timer expired because data or feedback packets were dropped in the network. In this case, the nofeedback timer is the backup mechanism for the sender to detect these losses, similar to the retransmit timer in TCP.

Note that when the sender stops sending data for a period of time, the receiver will stop sending feedback. When the sender's nofeedback timer expires, the sender could use the procedure above to limit the sending rate. If the sender subsequently starts to send again, X_recv_set will be used to limit the transmit rate, and slow-start behavior will occur until the transmit rate reaches X_Bps.

The TFRC sender's reduction of the allowed sending rate after the nofeedback timer expires is similar to TCP's reduction of the congestion window, cwnd, after each RTO seconds of an idle period, for TCP with Congestion Window Validation [RFC2861].

4.5. Reducing Oscillations

To reduce oscillations in queueing delay and sending rate in environments with a low degree of statistical multiplexing at the congested link, it is RECOMMENDED that the sender reduce the transmit rate as the queueing delay (and hence RTT) increases. To do this, the sender maintains R_sqmean, a long-term estimate of the square root of the RTT, and modifies its sending rate depending on how the square root of R_sample, the most recent sample of the RTT, differs from the long-term estimate. The long-term estimate R_sqmean 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, the constant used in the round-trip time estimate R. A value of 0.9 as the default for q2 is RECOMMENDED.

When sqrt(R_sample) is greater than R_sqmean, then the current round-trip time is greater than the long-term average, implying that queueing delay is probably increasing. In this case, the transmit rate is decreased to minimize oscillations in queueing delay.

The sender obtains the base allowed transmit rate, X, as described in step (4) of Section 4.3 above. It then calculates a modified instantaneous transmit rate X_inst, as follows:

```
X_inst = X * R_sqmean / sqrt(R_sample);
If (X_inst < s/t_mbi)
    X_inst = s/t_mbi;
```

Because we are using square roots, there is generally only a moderate difference between the instantaneous transmit rate X_inst and the allowed transmit rate X. For example, in a somewhat extreme case when the current RTT sample R_sample is twice as large as the long-term average, then sqrt(R_sample) will be roughly 1.44 times R_sqmean, and the allowed transmit rate will be reduced by a factor of roughly 0.7.

We note that this modification for reducing oscillatory behavior is not always needed, especially if the degree of statistical multiplexing in the network is high. We also note that the modification for reducing oscillatory behavior could cause problems for connections where the round-trip time is not strongly correlated with the queueing delay (e.g., in some wireless links, over paths with frequent routing changes, etc.). However, this modification SHOULD be implemented because it makes TFRC behave better in some environments with a low level of statistical multiplexing. The performance of this modification is illustrated in Section 3.1.3 of [FHPW00]. If it is not implemented, implementations SHOULD use a very low value of the weight q for the average round-trip time.

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 coarse-grain or irregular scheduling of the operating system. To help maintain the correct average sending rate, the TFRC sender MAY send some packets before their nominal send time.

In addition, the scheduling of packet transmissions controls the allowed burstiness of senders after an idle or data-limited period. The TFRC sender MAY accumulate sending 'credits' for past unused send times; this allows the TFRC sender to send a burst of data after an idle or data-limited period. To compare with TCP, TCP may send up to a round-trip time's worth of packets in a single burst, but never more. As examples, packet bursts can be sent by TCP when an ACK arrives acknowledging a window of data, or when a data-limited sender suddenly has a window of data to send after a delay of nearly a round-trip time.

To limit burstiness, a TFRC implementation MUST prevent bursts of arbitrary size. This limit MUST be less than or equal to one round-trip time's worth of packets. A TFRC implementation MAY limit bursts to less than a round-trip time's worth of packets. In addition, a TFRC implementation MAY use rate-based pacing to smooth bursts.

As an implementation-specific example, a sending loop could calculate the correct inter-packet interval, t_ipi, as follows:

```
t_ipi = s/X_inst;
```

Let t_now be the current time and i be a natural number, i = 0, 1, ..., with t_i the nominal send time for the i-th packet. Then, the nominal send time t_(i+1) would derive recursively as:

```
t_0 = t_now,
t_(i+1) = t_i + t_ipi.
```

For TFRC senders allowed to accumulate sending credits for unused send time over the last T seconds, the sender would be allowed to use unused nominal send times t_j for t_j < now - T, for T set to the round-trip time.

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. If the receiver has not yet detected a lost or marked packet, then the receiver does not calculate the loss event rate, but reports a loss event rate of zero.

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, it is REQUIRED 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 this 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 NDUPACK packets with a higher sequence number than the lost packet, for NDUPACK set to 3. The requirement for NDUPACK 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 NDUPACK 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 NDUPACK subsequent packets adaptive based on experienced packet reordering, but such a mechanism is not part of the current specification.

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.

If an ECN-marked packet is preceded by a possibly-lost packet, then the first detected congestion event begins with the lost packet. For example, if the receiver receives a data packet with sequence number n-1, followed by an unmarked data packet with sequence number n+1, and a marked data packet with sequence number n+2, then the receiver detects a congestion event when it receives the marked packet n+2. The first congestion event detected begins with the lost packet n. The guidelines in Section 5.2 below are used to determine whether the lost and marked packets belong to the same loss event or to separate loss events.

5.2. Translation from Loss History to Loss Events

TFRC requires that the loss fraction be robust to several consecutive packets lost or marked in 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 or marked 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 it is RECOMMENDED to use 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, before any packet arrivals with a sequence number above S_loss, with a sequence number below S_loss.

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

S_max is the largest sequence number.

Therefore, S_before < S_loss < S_after <= S_max.

T_loss is the nominal estimated arrival time for the lost packet.

T_before is the reception time of S_before.

T_after is the reception time of S_after.

Note that T_before < T_after.

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) );
```

To address sequence number wrapping, let S_MAX = 2^b, where b is the bit-length of sequence numbers in a given implementation. In this case, we can interpolate the arrival time T_loss as follows:

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

where

```
Dist(S_A, S_B) = (S_A + S_MAX - S_B) % S_MAX
```

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. The Size of a Loss Interval

After the detection of the first loss event, the receiver divides the sequence space into loss intervals. 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). Thus, loss interval A contains all of the packets transmitted by the sender starting with the first packet transmitted in loss interval A and ending with but not including the first packet transmitted in loss interval B.

The current loss interval I_0 is defined as the loss interval containing the most recent loss event. If that loss event started with packet sequence number S_A, and S_C is the highest received sequence number so far, then the size of I_0 is S_C - S_A + 1. As an example, if the current loss interval consists of a single ECN-marked packet, then S_A == S_C, and the size of the loss interval is one.

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. If the receiver has not yet seen a lost or marked packet, then the receiver does not calculate the average loss interval.

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

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

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 TFRC's speed in responding to changes in the level of congestion. It is RECOMMENDED to set the value n to 8. TFRC SHOULD NOT use values of n 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 current loss interval. We only include the current loss interval if it is sufficiently large to increase the average loss interval.

Let the most recent loss intervals be I_0 to I_k, where I_0 is the current loss interval. If there have been at least n loss intervals, then k is set to n; otherwise, k is the maximum number of loss intervals seen so far. We calculate the average loss interval I_mean as follows:

```
I_tot0 = 0;
I_tot1 = 0;
W_tot = 0;
for (i = 0 to k-1) {
    I_tot0 = I_tot0 + (I_i * w_i);
    W_tot = W_tot + w_i;
}
for (i = 1 to k) {
    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, when there have been at least n loss intervals, 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 [FHPW00a] and [W00], 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 = 0 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 current loss interval I_0. 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 size of current loss interval. 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 0.25. 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 respond 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 sent 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.

If the receiver was sending k feedback packets per RTT, for k>1, step (4) of Section 6.2 would be modified to set the feedback timer to expire after R_m/k seconds. However, each feedback packet would still report the receiver rate over the last RTT, not over a fraction of an RTT. In this document, we do not specify the modifications that might be required for a receiver sending more than one feedback packet per RTT. We note that 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) Check if done: If the new packet results in the detection of a new loss event, or if no feedback packet was sent when the feedback timer last expired, go to step 3. Otherwise, no action need be performed (unless the optimization in the next paragraph is used), so exit the procedure.

   An OPTIONAL 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.

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

4) Expire feedback timer: If p > p_prev, cause the feedback timer to expire and perform the actions described in Section 6.2.

   If p <= p_prev and no feedback packet was sent when the feedback timer last expired, cause the feedback timer to expire and perform the actions described in Section 6.2. If p <= p_prev and a feedback packet was sent when the feedback timer last expired, no action need be performed.

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.

For the m-th expiration of the feedback timer, 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. As described in Section 3.2.1, R_m is the sender's most recent estimate of the round-trip time, as reported in data packets. If data packets have been received since the previous feedback was sent, the receiver performs the following steps:

1) Calculate the average loss event rate using the algorithm described in Section 5.

2) Calculate the measured receive rate, X_recv, based on the packets received within the previous R_(m-1) seconds. This is performed whether the feedback timer expired at its normal time or expired early due to a new lost or marked packet (i.e., step (3) in Section 6.1).

   In the typical case, when the receiver is sending only one feedback packet per round-trip time and the feedback timer did not expire early due to a new lost packet, then the time interval since the feedback timer last expired would be R_(m-1) seconds.

   We note that when the feedback timer expires early due to a new lost or marked packet, the time interval since the feedback timer last expired is likely to be smaller than R_(m-1) seconds.

   For ease of implementation, if the time interval since the feedback timer last expired is not R_(m-1) seconds, the receive rate MAY be calculated over a longer time interval, the time interval going back to the most recent feedback timer expiration that was at least R_(m-1) seconds ago.

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.

Note that rule 2) above gives a minimum value for the measured receive rate X_recv of one packet per round-trip time. If the sender is limited to a sending rate of less than one packet per round-trip time, this will be due to the loss event rate, not from a limit imposed by the measured receive rate at the receiver.

If no data packets have been received since the last feedback was sent, then 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 data 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.

o Set the feedback timer to expire after R_i seconds.

If the first data packet does not contain an estimate R_i of the round-trip time, then the receiver sends a feedback packet for every arriving data packet until a data packet arrives containing an estimate of the round-trip time.

If the sender is using a coarse-grained timestamp that increments every quarter of a round-trip time, then a feedback timer is not needed, and the following procedure from RFC 4342 is used to determine when to send feedback messages.

o Whenever the receiver sends a feedback message, the receiver sets a local variable last_counter to the greatest received value of the window counter since the last feedback message was sent, if any data packets have been received since the last feedback message was sent.

o If the receiver receives a data packet with a window counter value greater than or equal to last_counter + 4, then the receiver sends a new feedback packet. ("Greater" and "greatest" are measured in circular window counter space.)

6.3.1. Initializing the Loss History after the First Loss Event

This section describes the procedure that MUST be used for initializing the loss history after the first loss event.

The number of packets until the first loss cannot be used to compute the allowed 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 maximum sending rate before the loss occurred. TFRC approximates this target rate, X_target, by the maximum value of X_recv so far. (For slow-start, for a particular round-trip time, the sender's sending rate is generally twice the receiver's receive rate for data sent over the previous 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_target, 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_target, given the round-trip time R, and the first loss interval is then set to 1/p. If the receiver knows the segment size, s, used by the sender, then the receiver MAY use the throughput equation for X; otherwise, the receiver MAY measure the receive rate in packets per second instead of bytes per second for this purpose, and use the throughput equation for X_pps. (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.)

Special care is needed for initializing the first loss interval when the first data packet is lost or marked. When the first data packet is lost in TCP, the TCP sender retransmits the packet after the retransmit timer expires. If TCP's first data packet is ECN-marked, the TCP sender resets the retransmit timer, and sends a new data packet only when the retransmit timer expires [RFC3168] (Section 6.1.2). For TFRC, if the first data packet is lost or ECN-marked, then the first loss interval consists of the null interval with no data packets. In this case, the loss interval length for this (null) loss interval SHOULD be set to give a similar sending rate to that of TCP, as specified in the paragraph below.

When the first TFRC loss interval is null, meaning that the first data packet is lost or ECN-marked, in order to follow the behavior of TCP, TFRC wants the allowed sending rate to be 1 packet every two round-trip times, or equivalently, 0.5 packets per RTT. Thus, the TFRC receiver calculates the loss interval that would be required to produce the target rate X_target of 0.5/R packets per second, for the round-trip time R, and uses this synthetic loss interval for the first loss interval. The TFRC receiver uses 0.5/R packets per second as the minimum value for X_target when initializing the first loss interval.

We note that even though the TFRC receiver reports a synthetic loss interval after the first loss event, the TFRC receiver still reports the measured receive rate X_recv, as specified in Section 6.2 above.

7. Sender-Based Variants

In a sender-based variant of TFRC, 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.

The main advantage of a sender-based variant of TFRC is that the sender does 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 where it is desirable to offload work from the server to the client as much as possible.

RFC 4340 and RFC 4342 together specify DCCP's CCID 3, which can be used as a sender-based variant of TFRC. In CCID 3, each feedback packet from the receiver contains a Loss Intervals option, reporting the lengths of the most recent loss intervals. Feedback packets may also include the Ack Vector option, allowing the sender to determine exactly which packets were dropped or marked and to check the information reported in the Loss Intervals options. The Ack Vector option can also include ECN Nonce Echoes, allowing the sender to verify the receiver's report of having received an unmarked data packet. The Ack Vector option allows the sender to see for itself which data packets were lost or ECN-marked, to determine loss intervals, and to calculate the loss event rate. Section 9 of RFC 4342 discusses issues in the sender verifying information reported by the receiver.

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 implementation issues.

8.1. Computing the Throughput Equation

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

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

for

$$f(p) = sqrt(2*p/3) + (12*sqrt(3*p/8) * p * (1+32*p^2)).$$

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.

8.2. Sender Behavior When a Feedback Packet Is Received

This section discusses implementation issues for sender behavior when a feedback packet is received, from Section 4.3.

8.2.1. Determining If an Interval Was a Data-Limited Interval

When a feedback packet is received, the sender has to determine if the entire interval covered by that feedback packet was a data-limited period. This section discusses one possible implementation for the sender to determine if the interval covered by a feedback packet was a data-limited period.

If the feedback packets all report the timestamp of the last data packet received, then let t_new be the timestamp reported by this feedback packet. Because all feedback packets cover an interval of at least a round-trip time, it is sufficient for the sender to determine if there was any time in the period (t_old, t_new] when the sender was not data-limited, for R the sender's estimate of the round-trip time, and for t_old set to t_new - R. (This procedure estimates the interval covered by the feedback packet, rather than computing it exactly. This seems fine to us.)

The pseudocode for determining if the sender was data-limited over the entire interval covered in a feedback packet is given below. The variables NotLimited1 and NotLimited2 both represent times when the sender was *not* data-limited.

```
Initialization:
    NotLimited1 = NotLimited2 = t_new = t_next = 0;
    t_now = current time;

After sending a segment:
    If (sender has sent all it is allowed to send) {
        // Sender is not data-limited at this instant.
        If NotLimited1 <= t_new
            // Goal: NotLimited1 > t_new.
            NotLimited1 = t_now;
        Else if (NotLimited2 <= t_next)
            // Goal: NotLimited2 > t_next.
            NotLimited2 = t_now;
    }
```

```
When a feedback packet is received, is this interval data-limited:
    t_new = timestamp reported in feedback packet.
    t_old = t_new - R.                         // local variable
    t_next = t_now;
    If ((t_old < NotLimited1 <= t_new) or
        (t_old < NotLimited2 <= t_new))
        This was not a data-limited interval;
    Else
        This was a data-limited interval.
    If (NotLimited1 <= t_new && NotLimited2 > t_new)
        NotLimited1 = NotLimited2;
```

Transmission times refer to transmission of a segment or segments to the layer below.

Between feedback packets, (t_old, t_new] gives the transmission time interval estimated to be covered by the most recent feedback packet, and t_next gives a time at least a round-trip time greater than t_new. The next feedback packet can be expected to cover roughly the interval (t_new, t_next] (unless the receiver sends the feedback packet early because it is reporting a new loss event). The goal is for NotLimited1 to save a non-data-limited time in (t_new, t_next], if there was one, and for NotLimited2 to save a non-data-limited time after t_next.

When a feedback packet was received, if either NotLimited1 or NotLimited2 is in the time interval covered by the feedback packet, then the interval is not a data-limited interval; the sender was not data-limited at least once during that time interval. If neither NotLimited1 nor NotLimited2 is in the time interval covered by a feedback packet, then the sender is assumed to have been data-limited over that time interval.

We note that this procedure is a heuristic, and in some cases the sender might not determine correctly if the sender was data-limited over the entire interval covered by the feedback packet. This heuristic does not address the possible complications of reordering.

That seems acceptable to us. In order to improve its accuracy in identifying if the entire interval covered by a feedback packet was a data-limited interval, the sender could save more NotLimited times.

In some implementations of TFRC, the sender sends coarse-grained timestamps that increment every quarter of a round-trip time, and the feedback packet reports the greatest valid sequence number received so far instead of reporting the timestamp of the last packet received. In this case, the sender can maintain per-packet state to determine t_new (the time that the acknowledged packet was sent), or the sender can estimate t_new from its estimate of the round-trip time and the elapsed time t_delay reported by the feedback packet.

8.2.2. Maintaining X_recv_set

To reduce the complexity of maintaining X_recv_set, it is sufficient to limit X_recv_set to at most N=3 elements. In this case, the procedure Update X_recv_set() would be modified as follows:

```
Update X_recv_set():
    Add X_recv to X_recv_set;
    Delete from X_recv_set values older than
        two round-trip times.
    Keep only the most recent N values.
```

Maintaining at most *two* elements in X_recv_set would be sufficient for the sender to save an old value of X_recv from before a data-limited period, and to allow the sender not to be limited by the first feedback packet after an idle period (reporting a receive rate of one packet per round-trip time). However, it is *possible* that maintaining at most two elements in X_recv_set would not give quite as good performance as maintaining at most three elements. Maintaining three elements in X_recv_set would allow X_recv_set to contain X_recv values from two successive feedback packets, plus a more recent X_recv value from a loss event.

8.3. Sending Packets before Their Nominal Send Time

This section discusses one possible scheduling mechanism for a sender in an operating system with a coarse-grained timing granularity (from Section 4.6).

Let t_gran be the scheduling timer granularity of the operating system. Let t_ipi be the inter-packet interval, as specified in Section 4.6. If the operating system has a coarse timer granularity or otherwise cannot support short t_ipi intervals, then either the TFRC sender will be restricted to a sending rate of at most 1 packet every t_gran seconds, or the TFRC sender must be allowed to send short bursts of packets. In addition to allowing the sender to accumulate sending credits for past unused send times, it can be useful to allow the sender to send a packet before its scheduled send time, as described in the section below.

A parameter, t_delta, may be used to allow a packet to be sent before its nominal send time. Consider an application that becomes idle and requests re-scheduling for time t_i = t_(i-1) + t_ipi, for t_(i-1) the send time for the previous packet. When the application is rescheduled, it checks the current time, t_now. If (t_now > t_i - t_delta), then packet i is sent. When the nominal send time, t_i, of the next packet is calculated, it may already be the case that t_now > t_i - t_delta. In such a case, the packet would be sent immediately.

In order to send at most one packet before its nominal send time, and never to send a packet more than a round-trip time before its nominal send time, the parameter t_delta would be set as follows:

```
t_delta = min(t_ipi, t_gran, rtt)/2;
```

(The scheduling granularity t_gran is 10 ms on some older Unix systems.)

As an example, consider a TFRC flow with an allowed sending rate X of 10 packets per round-trip time (PPR), a round-trip time of 100 ms, a system with a scheduling granularity t_gran of 10 ms, and the ability to accumulate unused sending credits for a round-trip time. In this case, t_ipi is 1 ms. The TFRC sender would be allowed to send packets 0.5 ms before their nominal sending time, and would be allowed to save unused sending credits for 100 ms. The scheduling granularity of 10 ms would not significantly affect the performance of the connection.

As a different example, consider a TFRC flow with a scheduling granularity greater than the round-trip time, for example, with a round-trip time of 0.1 ms and a system with a scheduling granularity of 1 ms, and with the ability to accumulate unused sending credits for a round-trip time. The TFRC sender would be allowed to save unused sending credits for 0.1 ms. If the scheduling granularity *did not* affect the sender's response to an incoming feedback packet, then the TFRC sender would be able to send an RTT of data (as determined by the allowed sending rate) each RTT, in response to incoming feedback packets. In this case, the coarse scheduling granularity would not significantly reduce the sending rate, but the sending rate would be bursty, with a round-trip time of data sent in response to each feedback packet.

However, performance would be different, in this case, if the operating system scheduling granularity affected the sender's response to feedback packets as well as the general scheduling of the sender. In this case, the sender's performance would be severely limited by the scheduling granularity being greater than the round-trip time, with the sender able to send an RTT of data, at the allowed sending rate, at most once every 1 ms. This restriction of the sending rate is an unavoidable consequence of allowing burstiness of at most a round-trip time of data.

8.4. Calculation of the Average Loss Interval

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.

8.5. The Optional History Discounting Mechanism

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. Changes from RFC 3448

9.1. Overview of Changes

This section summarizes the changes from RFC 3448. At a high level, the main change is to add mechanisms to address the case of a data-limited sender. This document also explicitly allows the TFRC sender to accumulate up to a round-trip time of unused send credits, and as a result to send a burst of packets if data arrives from the application in a burst after a data-limited period. This issue was not explicitly addressed in RFC 3448.

This document changes RFC 3448 to incorporate TCP's higher initial sending rates from RFC 3390. This document also changes RFC 3448 to allow RFC 4342's use of a coarse-grained timestamp on data packets instead of a more fine-grained timestamp.

Other changes address corner cases involving slow-start, the response when the first data packet is dropped, and the like. This document also incorporates the items in the RFC 3448 Errata.

This section is non-normative; the normative text is in the cited sections.

9.2. Changes in Each Section

Section 4.1, estimating the average segment size: Section 4.1 was modified to give a specific algorithm that could be used for estimating the average segment size.

Section 4.2, update to the initial sending rate: In RFC 3448, the initial sending rate was two packets per round-trip time. In this document, the initial sending rate can be as high as four packets per round-trip time, following RFC 3390. The initial sending rate was changed to be in terms of the segment size s, not in terms of the MSS.

Section 4.2 now says that tld, the Time Last Doubled during slow-start, can be initialized to either 0 or to -1. Section 4.2 was also clarified to say that RTT measurements do not only come from feedback packets; they could also come from other places, such as the SYN exchange.

Section 4.3, response to feedback packets: Section 4.3 was modified to change the way that the receive rate is used in limiting the sender's allowed sending rate, by using the set of receive rate values of the last two round-trip times, and initializing the set of receive rate values by a large value.

The larger initial sending rate in Section 4.2 is of little use if the receiver sends a feedback packet after the first packet is received, and the sender, in response, reduces the allowed sending rate to at most two packets per RTT, which would be twice the receive rate. Because of the change in the sender's processing of the receive rate, the sender now does not reduce the allowed sending rate to twice the reported receive rate in response to the first feedback packet.

During a data-limited period, the sender saves the receive rate reported from just before the data-limited period, if it is larger than the receive rate during the data-limited period. The sender also reduces the saved values in X_recv_set in response to a loss during a data-limited period. Appendix C discusses this response further.

Section 4.4, response to an idle period: Following Section 5.1 from [RFC4342], this document specifies that when the sending rate is reduced after an idle period that covers the period since the nofeedback timer was set, the allowed sending rate is not reduced below the initial sending rate. (In Section 4.4, the variable recover_rate is set to the initial sending rate.)

Section 4.4, correction from [RFC3448Err]. RFC 3448 had contradictory text about whether the sender halved its sending rate after *two* round-trip times without receiving a feedback report, or after *four* round-trip times. This document clarifies that the sender halves its sending rate after four round-trip times without receiving a feedback report [RFC3448Err].

Section 4.4, clarification for slow-start: Section 4.4 was clarified to specify that on the expiration of the nofeedback timer, if p = 0, X_Bps cannot be used, because the sender does not yet have a value for X_Bps. Section 4.4 was also clarified to check the case when the sender does not yet have an RTT sample, but has sent a packet since the nofeedback timer was set.

Section 4.6: credits for unused send time:

Section 4.6 has been clarified to say that the TFRC sender gets to accumulate up to an RTT of credits for unused send time. Section 4.6 was also rewritten to clarify what is specification and what is implementation.

Section 5.4, clarification: Section 5.4 was modified to clarify the receiver's calculation of the average loss interval when the receiver has not yet seen n loss intervals.

Section 5.5, correction: Section 5.5 was corrected to say that the loss interval I_0 includes all transmitted packets, including lost and marked packets (as defined in Section 5.3 in the general definition of loss intervals).

Section 5.5, correction from [RFC3448Err]: A line in Section 5.5 was changed from

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

to

```
for (i = 0 to n) { DF_i = 1; }
```

[RFC3448Err].

Section 5.5, history discounting: THRESHOLD, the lower bound on the history discounting parameter DF, has been changed from 0.5 to 0.25, to allow more history discounting when the current interval is long.

Section 6, multiple feedback packets: Section 6 now contains more discussion of procedures if the receiver sends multiple feedback packets each round-trip time.

Section 6.3, initialization of the feedback timer: Section 6.3 now specifies the receiver's initialization of the feedback timer if the first data packet received does not have an estimate of the round-trip time.

Section 6.3, a coarse-grained timestamp: Section 6.3 was modified to incorporate, as an option, a coarse-grained timestamp from the sender that increments every quarter of a round-trip time, instead of a more fine-grained timestamp. This follows RFC 4342.

Section 6.3.1, after the first loss event: Section 6.3.1 now says that for initializing the loss history after the first loss event, the receiver uses the maximum receive rate so far, instead of the receive rate in the last round-trip time.

Section 6.3.1, if the first data packet is dropped: Section 6.3.1 now contains a specification for initializing the loss history if the first data packet sent is lost or ECN-marked.

Section 7, sender-based variants: Section 7's discussion of sender-based variants has been expanded, with reference to RFC 4342.

10. 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 [RFC3540]. 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.

10.1. Security Considerations for TFRC in DCCP

TFRC is currently used in Congestion Control ID 3 (CCID 3) [RFC4342] of the Datagram Congestion Control Protocol (DCCP) [RFC4340]. The Security Considerations section of RFC 4340 [RFC4340] (Section 18) discusses some of the security issues of DCCP, including sequence number validity checks to protect against hijacked connections. Section 18 of RFC 4340 also discusses mechanisms in DCCP to limit the potential impact of denial-of-service attacks.

RFC 4342 specifies the use of TFRC in CCID 3. RFC 4342 includes extensive discussions of the mechanisms the sender can use to verify the information sent by the receiver. When ECN is used with CCID 3, the receiver returns ECN Nonce information to the sender, to allow the sender to verify information sent by the receiver. When ECN is not used, Section 9 of RFC 4342 discusses how the sender could still use various techniques that might catch the receiver in an error in reporting congestion. However, as stated in RFC 4342, this is not as robust or non-intrusive as the verification provided by the ECN Nonce.

11. 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 Dado Colussi, Gorry Fairhurst, Ladan Gharai, Wim Heirman, Eddie Kohler, Ken Lofgren, Mike Luby, Ian McDonald, Vladimir Moltchanov, Colin Perkins, Michele R., Gerrit Renker, Arjuna Sathiaseelan, Vladica Stanisic, Randall Stewart, Eduardo Urzaiz, 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 [RFC3448] to produce a working implementation.

Appendix A. Terminology

This document uses the following terms. Timer variables (e.g., t_now, tld) are assumed to be in seconds, with a timer resolution of at least a millisecond.

data-limited interval:
An interval where the sender is data-limited (not sending as much as it is allowed to send) over the entire interval (Section 4.3).

DF: Discount factor for a loss interval (Section 5.5).

initial_rate:
Allowed initial sending rate.

last_counter:
Greatest received value of the window counter (Section 6.3).

n: Number of loss intervals.

NDUPACK:
Number of dupacks for inferring loss (constant) (Section 5.1).

nofeedback timer:
Sender-side timer (Section 4).

p: Estimated Loss Event Rate.

p_prev:
Previous value of p (Section 6.1).

q: Filter constant for RTT (constant) (Section 4.3).

q2: Filter constant for long-term RTT (constant) (Section 4.6).

R: Estimated path round-trip time.

R_m:
A specific estimate of the path round-trip time (Sections 4.3, 6).

R_sample:
Measured path RTT (Section 4.3).

R_sqmean:
Long-term estimate of the square root of the RTT (Section 4.6).

recover_rate:
Allowed rate for resuming after an idle period (Section 4.4).

recv_limit;
   Limit on sending rate computed from X_recv_set (Section 4.3).

s: Nominal packet size in bytes.

S: Sequence number.

t_delay:
   Reported time delay between receipt of the last packet at the receiver and the generation of the feedback packet (Section 3.2.2).

t_delta:
   Parameter for flexibility in send time (Section 8.3).

t_gran:
   Scheduling timer granularity of the operating system (constant) (Section 8.3).

t_ipi:
   Inter-packet interval for sending packets (Section 4.6).

t_mbi:
   Maximum RTO value of TCP (constant) (Section 4.3).

t_recvdata:
   Timestamp of the last data packet received (Section 3.2.2).

timer_limit:
   Limit on the sending rate from the expiration of the nofeedback timer (Section 4.4).

tld:
   Time Last Doubled (Section 4.2).

t_now:
   Current time (Section 4.3).

t_RTO:
   Estimated RTO of TCP (Section 4.3).

X: Allowed transmit rate, as limited by the receive rate.

X_Bps:
   Calculated sending rate in bytes per second (Section 3.1).

X_pps:
   Calculated sending rate in packets per second (Section 3.1).

X_inst:
Instantaneous allowed transmit rate (Section 4.6).

X_recv:
Estimated receive rate at the receiver (Section 3.2.2).

X_recv_set:
A small set of recent X_recv values (Section 4.3).

X_target:
The target sending rate after the first loss event (Section 6.3.1).

W_init:
TCP initial window (constant) (Section 4.2).

Appendix B. The Initial Value of the Nofeedback Timer

Why is the initial value of TFRC's nofeedback timer set to two seconds, instead of the recommended initial value of three seconds for TCP's retransmit timer, from [RFC2988]? There is not any particular reason why TFRC's nofeedback timer should have the same initial value as TCP's retransmit timer. TCP's retransmit timer is used not only to reduce the sending rate in response to congestion, but also to retransmit a packet that is assumed to have been dropped in the network. In contrast, TFRC's nofeedback timer is only used to reduce the allowed sending rate, not to trigger the sending of a new packet. As a result, there is no danger to the network for the initial value of TFRC's nofeedback timer to be smaller than the recommended initial value for TCP's retransmit timer.

Further, when the nofeedback timer has not yet expired, TFRC has a more slowly responding congestion control mechanism than TCP, and TFRC's use of the receive rate for limiting the sending rate is somewhat less precise than TCP's use of windows and ack-clocking, so the nofeedback timer is a particularly important safety mechanism for TFRC. For all of these reasons, it is perfectly reasonable for TFRC's nofeedback timer to have a smaller initial value than that of TCP's retransmit timer.

Appendix C. Response to Idle or Data-Limited Periods

Future work could explore alternate responses to using the receive rate during a data-limited period, and to responding to a loss event during a data-limited period.

In particular, an Experimental RFC [RFC2861] specifies Congestion Window Validation (CWV) for TCP. For this discussion, we use the term "Standard TCP" to refer to the TCP congestion control mechanisms in [RFC2581] and [RFC2581bis]. [RFC2861] specifies a different response to idle or data-limited periods than those of Standard TCP. With CWV, the TCP sender halves the congestion window after each RTO during an idle period, down to the initial window. Similarly, with CWV the TCP sender halves the congestion window half-way down to the flight size after each RTO during a data-limited period.

This document already specifies a TFRC response to idle periods that is similar to that of TCP with Congestion Window Validation. However, this document does not specify a TFRC response to data-limited periods similar to that of CWV. Adding such a mechanism to TFRC would require a one-line change to step (4) of Section 4.3. In particular, the sender's response to a feedback packet could be changed from:

```
If (the entire interval covered by the feedback packet
      was a data-limited interval) {
    If (the feedback packet reports a new loss event or an
                 increase in the loss event rate p) {
        Halve entries in X_recv_set;
        X_recv = 0.85 * X_recv;
        Maximize X_recv_set();
        recv_limit = max (X_recv_set);
    } Else {
        Maximize X_recv_set();
        recv_limit = 2 * max (X_recv_set);
    }
}
```

to:

```
If (the entire interval covered by the feedback packet
      was a data-limited interval) {
    Multiply old entries in X_recv_set by 0.85;
    If (the feedback packet reports a new loss event or an
                 increase in the loss event rate p) {
        Multiply new value X_recv by 0.85.
    }
    Maximize X_recv_set();
    recv_limit = 2 * max (X_recv_set);
}
```

In particular, if the receive rate from before a data-limited period is saved in X_recv_set, then the change in step (4) above would multiply that receive rate by 0.85 each time that a feedback packet is received and the above code is executed. As a result, after four successive round-trip times of data-limited intervals, the receive rate from before the data-limited period would be reduced by 0.85^4 = 0.52. Thus, this one-line change to step (4) of Section 4.3 would result in the allowed sending rate being halved for each four roundtrip times in which the sender was data-limited. Because of the nature of X_recv_set, this mechanism would never reduce the allowed sending rate below twice the most recent receive rate.

We note that in the suggested code above, with CWV-style behavior in response to data-limited intervals, we keep

```
recv_limit = 2 * max (X_recv_set);
```

instead of using

```
recv_limit = max (X_recv_set);
```

following loss events in data-limited intervals. This relaxed response to a loss event is allowed because the CWV-style behavior itself limits rapid fluctuations in the sending rate during data-limited periods.

C.1. Long Idle or Data-Limited Periods

Table 1 summarizes the response of Standard TCP [RFC2581], TCP with Congestion Window Validation [RFC2861], Standard TFRC [RFC3448], and Revised TFRC (this document) in response to long idle or data-limited periods. For the purposes of this section, we define a long period as a period of at least an RTO.

Protocol	Long idle periods	Long data-limited periods
Standard TCP:	Window -> initial.	Window increases for each cwnd of data.
TCP with CWV:	Halve window (not below initial cwnd).	Reduce window half way to used window.
Standard TFRC:	Halve rate (not below 2 pkts/rtt). One RTT after sending pkt, rate is limited by X_recv.	Rate limited to twice receive rate.
Revised TFRC:	Halve rate (not below initial rate).	Rate limited to twice max (current X_recv, receive rate before data-limited period).

Table 1: Response to Long Idle or Data-Limited Periods

Standard TCP after long idle periods: For Standard TCP, [RFC2581] specifies that TCP SHOULD set the congestion window to no more than the initial window after an idle period of at least an RTO. (To be precise, RFC 2581 specifies that the TCP sender should set cwnd to the initial window if the sender has not sent data in an interval exceeding the retransmission timeout.)

Standard TCP after long data-limited periods: Standard TCP [RFC2581] does not reduce TCP's congestion window after a data-limited period, when the congestion window is not fully used. Standard TCP in [RFC2581] uses the FlightSize, the amount of outstanding data in the network, only in setting the slow-start threshold after a retransmit timeout. Standard TCP is not limited by TCP's ack-clocking mechanism during a data-limited period.

Standard TCP's lax response to a data-limited period is quite different from its stringent response to an idle period.

TCP with Congestion Window Validation (CWV) after long idle periods: As an experimental alternative, [RFC2861] specifies a more moderate response to an idle period than that of Standard TCP, where during an idle period the TCP sender halves cwnd after each RTO, down to the initial cwnd.

TCP with Congestion Window Validation after long data-limited periods: As an experimental alternative, [RFC2861] specifies a more stringent response to a data-limited period than that of Standard TCP, where after each RTO seconds of a data-limited period, the congestion window is reduced half way down to the window that is actually used.

The response of TCP with CWV to an idle period is similar to its response to a data-limited period. TCP with CWV is less restrictive than Standard TCP in response to an idle period, and more restrictive than Standard TCP in response to a data-limited period.

Standard TFRC after long idle periods: For Standard TFRC, [RFC3448] specifies that the allowed sending rate is halved after each RTO seconds of an idle period. The allowed sending rate is not reduced below two packets per RTT after idle periods. After an idle period, the first feedback packet received reports a receive rate of one packet per round-trip time, and this receive rate is used to limit the sending rate. Standard TFRC effectively slow-starts up from this allowed sending rate.

Standard TFRC after long data-limited periods: [RFC3448] does not distinguish between data-limited and non-data-limited periods. As a consequence, the allowed sending rate is limited to at most twice the receive rate during and after a data-limited period. This is a very restrictive response, more restrictive than that of either Standard TCP or of TCP with CWV.

Revised TFRC after long idle periods: For Revised TFRC, this document specifies that the allowed sending rate is halved after each RTO seconds of an idle period. The allowed sending rate is not reduced below the initial sending rate as the result of an idle period. The first feedback packet received after the idle period reports a receive rate of one packet per round-trip time. However, the Revised TFRC sender does not use this receive rate for limiting the sending rate. Thus, Revised TFRC differs from Standard TFRC in the lower limit used in the reduction of the sending rate, and in the better response to the first feedback packet received after the idle period.

Revised TFRC after long data-limited periods: For Revised TFRC, this document distinguishes between data-limited and non-data-limited periods. As specified in Section 4.3, during a data-limited period Revised TFRC remembers the receive rate before the data-limited period began, and does not reduce the allowed sending rate below twice that receive rate. This is somewhat similar to the response of Standard TCP, and is quite different from the very restrictive response of Standard TFRC to a data-limited period. However, the response of Revised TFRC is not as conservative as the response of TCP with Congestion Window Validation, where the congestion window is gradually reduced down to the window actually used during a data-limited period.

We note that for current TCP implementations, the congestion window is generally not increased during a data-limited period (when the current congestion window is not being fully used) [MAF05] (Section 5.7). We note that there is no mechanism comparable to this in Revised TFRC.

Recovery after idle or data-limited periods: When TCP reduces the congestion window after an idle or data-utilized period, TCP can set the slow-start threshold, ssthresh, to allow the TCP sender to slow-start back up towards its old sending rate when the idle or data-limited period is over. However, in TFRC, even when the TFRC sender's sending rate is restricted by twice the previous receive rate, this results in the sender being able to double the sending rate from one round-trip time to the next, if permitted by the throughput equation. Thus, TFRC does not need a mechanism such as TCP's setting of ssthresh to allow a slow-start after an idle or data-limited period.

For future work, one avenue to explore would be the addition of Congestion Window Validation mechanisms for TFRC's response to data-limited periods. Currently, following Standard TCP, during data-limited periods Revised TFRC does not limit its allowed sending rate as a function of the receive rate.

C.2. Short Idle or Data-Limited Periods

Table 2 summarizes the response of Standard TCP [RFC2581], TCP with Congestion Window Validation [RFC2861], Standard TFRC [RFC3448], and Revised TFRC (this document) in response to short idle or data-limited periods. For the purposes of this section, we define a short period as a period of less than an RTT.

Protocol	Short idle periods	Short data-limited periods
Standard TCP:	Send a burst up to cwnd.	Send a burst up to cwnd.
TCP with CWV:	Send a burst up to cwnd.	Send a burst up to cwnd.
Standard TFRC:	?	?
Revised TFRC:	Send a burst (up to an RTT of unused send credits).	Send a burst (up to an RTT of unused send credits).

Table 2: Response to Short Idle or Data-Limited Periods

Table 2 shows that Revised TFRC has a similar response to that of Standard TCP and of TCP with CWV to a short idle or data-limited period. For a short idle or data-limited period, TCP is limited only by the size of the unused congestion window, and Revised TFRC is limited only by the number of unused send credits (up to an RTT's worth). For Standard TFRC, [RFC3448] did not explicitly specify the behavior with respect to unused send credits.

C.3. Moderate Idle or Data-Limited Periods

Table 3 summarizes the response of Standard TCP [RFC2581], TCP with Congestion Window Validation [RFC2861], Standard TFRC [RFC3448], and Revised TFRC (this document) in response to moderate idle or data-limited periods. For the purposes of this section, we define a moderate period as a period greater than an RTT, but less than an RTO.

Protocol	Moderate idle periods	Moderate data-limited periods
Standard TCP:	Send a burst up to cwnd.	Send a burst up to cwnd.
TCP with CWV:	Send a burst up to cwnd.	Send a burst up to cwnd.
Standard TFRC:	?	Limited by X_recv.
Revised TFRC:	Send a burst (up to an RTT of unused send credits).	Send a burst (up to an RTT of unused send credits).

Table 3: Response to Moderate Idle or Data-Limited Periods

Table 3 shows that Revised TFRC has a similar response to that of Standard TCP and of TCP with CWV to a moderate idle or data-limited period. For a moderate idle or data-limited period, TCP is limited only by the size of the unused congestion window. For a moderate idle period, Revised TFRC is limited only by the number of unused send credits (up to an RTT's worth). For a moderate data-limited period, Standard TFRC would be limited by X_recv from the most recent feedback packet. In contrast, Revised TFRC is not limited by the receive rate from data-limited periods that cover an entire feedback period of a round-trip time. For Standard TFRC, [RFC3448] did not explicitly specify the behavior with respect to unused send credits.

C.4. Losses During Data-Limited Periods

This section discusses the response to a loss during a data-limited period.

Protocol	Response to a loss during a data-limited period
Standard TCP:	Set ssthresh, cwnd to FlightSize/2.
TCP with CWV:	Same as Standard TCP.
Standard TFRC:	Calculate X_Bps, send at most 2*X_recv.
Revised TFRC:	Calculate X_Bps, send at most recv_limit. In addition, modify X_recv_set.

Table 4: Response to a Loss during a Data-Limited Period

In TCP [RFC2581], the response to a loss during a data-limited period is the same as the response to a loss at any other time in TCP. This response is to set the congestion window to half of the FlightSize, where the FlightSize is the actual amount of unacknowledged data. Thus, after a loss during a data-limited period, the TCP sender must halve its allowed sending rate, as it normally does in response to a loss.

In Standard TFRC, the response to a loss during a data-limited period is also the same as the response to a loss at any other time in Standard TFRC. The sending rate is limited by X_Bps, from the throughput equation, and the sending rate is also limited by twice X_recv, the most recent receive rate. As a result, after a loss in a data-limited period, the sender can at most double its sending rate to twice X_recv, even if the throughput equation X_Bps would allow a sending rate much higher than that.

In Revised TFRC, there have been changes to the use of the receive rate X_recv during data-limited intervals; the sender is limited to sending at most recv_limit, where the sender can remember the receive rate X_recv from just before the data-limited period. This allows the sender to more than double its sending rate during data-limited periods, up to the receive rate from before the data-limited period (if allowed by the throughput equation as given in X_Bps). This is similar to Standard TCP's practice of not reducing the window during data-limited periods (in the absence of loss).

As with Standard TFRC, during a data-limited period the Revised TFRC sender is sending less than is allowed by the throughput equation X_Bps. After the loss event, the sender still might not want to be sending as much as allowed by the recalculated value of X_Bps that takes into account the new loss event. Revised TFRC adds an additional mechanism to gradually limit the sender's sending rate after losses during data-limited periods. Unlike TCP's response of setting cwnd to half the FlightSize, this additional mechanism in Revised TFRC uses TFRC's practice of using slowly-responding changes for both increases and decreases in the allowed sending rate.

This is done in Revised TFRC (in step (4) of Section 4.3) by decreasing the entry in X_recv_set after a loss in a data-limited interval, and by allowing the sender to send at most max (X_recv_set), instead of at most twice max (X_recv_set), in the immediate round-trip time following the reported loss. Thus, the 'price' for allowing the sender to send more than twice the most immediately reported value of X_recv during a data-limited interval is the introduction of an additional mechanism to reduce this allowed sending rate following losses in data-limited periods.

In TFRC's response to a loss in a data-limited interval, we have considered the following examples.

Example 1, Losses *after* a Data-Limited Period: This example shows that losses after a data-limited period has ended are addressed by the throughput equation X_Bps.

```
-------------------------------------------------------------------
Stage 1: Not data-limited.
         Sending 100 packets per round-trip time (PPR).
Stage 2: Data-limited, sending 10 PPR.
Stage 3: Not data-limited.
         Sending 100 PPR again, as allowed by X_Bps.
         A packet loss in the first RTT of Stage 3.
         X_Bps is updated,
Response of Revised TFRC: a slight reduction in the allowed sending
  rate, depending on the number of packets since the last loss event.
-------------------------------------------------------------------
```

Table 5: Example 1, Losses after a Data-Limited Period

For example 1, when there is a packet loss in the first RTT of Stage 3, this will be reflected in a modified value of X_Bps, and future loss events would result in future reductions of the throughput equation X_Bps. In particular, following TFRC's standard use of the throughput equation [FHPW00] (Section A.2), the allowed TFRC sending rate would be halved after something like five successive round-trip times with loss.

Example 2, a Mildly Data-Limited Sender: This example considers losses in a data-limited period when, during the data-limited period, the sender is sending *almost* as much as it is allowed to send.

```
-------------------------------------------------------------------
Stage 1: Not data-limited.  Sending 100 PPR.
Stage 2: Data-limited, sending 99 PPR.
         A packet loss in Stage 2.
Response of Revised TFRC: a slight reduction in the allowed sending
  rate, down to 85 PPR or less, depending on the number of packets
  since the last loss event.
-------------------------------------------------------------------
```

Table 6: Example 2, a Mildly Data-Limited Sender

Consider a Revised TFRC connection where the sender has been sending a hundred PPR and then enters a data-limited period of sending only 99 PPR because of data limitations from the application. (That is, at every instance of time during the data-limited period, the sender could have sent one more packet.) If there are losses in the data-limited period, the allowed sending rate is reduced to min(X_Bps, recv_limit), where both the throughput equation X_Bps and the limit recv_limit force a slight reduction in the allowed sending rate.

Example 3, a Single Packet Loss during a Data-Limited Period. This example considers the loss of a single packet during a data-limited period, after the sender has not sent a packet for two RTTs.

```
-------------------------------------------------------------------
Stage 1: Not data-limited.  Sending 100 PPR.
Stage 2: Data-limited, sending 10 PPR.
Stage 3: Data-limited, sending no data for two RTTs.
Stage 4: Data-limited, sending one packet, which is ECN-marked.
Response of Revised TFRC: a reduction in the allowed sending
  rate, down to 50 PPR or less.  For each loss event during
  the data-limited period, the 'remembered' X_recv from before
  the data-limited period is effectively halved.
-------------------------------------------------------------------
```

Table 7: Example 3, a Single Packet Loss

Consider a Revised TFRC connection where the sender has been sending a hundred PPR, and then enters a data-limited period of sending only ten PPR, and then does not send any packets for two RTTs, and then sends a single packet, which is ECN-marked. In this case, with Revised TFRC, for each loss event during the data-limited period, the sender halves its 'remembered' X_recv from before the data-limited period

Example 4, Losses after Increasing the Sending Rate during a Data-Limited Period. This example considers losses when the sender significantly increases its sending rate during a data-limited period.

```
-------------------------------------------------------------------
Stage 1: Not data-limited.  Sending 100 PPR.
Stage 2: Data-limited, sending 1 PPR.
Stage 3: Data-limited, sending 20 PPR.
         Several packets are lost in each RTT of Stage 3.
         During Stage 3, the sender would *like* to send 20 PPR.
Response of Revised TFRC:  For each loss event during
  the data-limited period, the 'remembered' X_recv from before
  the data-limited period is effectively halved, and the most
  recent X_recv is reduced by 0.85.
-------------------------------------------------------------------
```

Table 8: Example 4, Losses after Increasing the Sending Rate

Consider a Revised TFRC connection where the sender has been sending a hundred PPR, and then enters a data-limited period of sending only one PPR, and then, while still data-limited, increases its sending rate to twenty PPR, where it experiences a number of successive loss events.

In this case, with Revised TFRC, for each loss event during the data-limited period, the sender halves its 'remembered' X_recv from before the data-limited period, and the most recent X_recv is reduced by 0.85.

C.5. Other Patterns

Other possible patterns to consider in evaluating Revised TFRC would be to compare the behavior of TCP, Standard TFRC, and Revised TFRC for connections with alternating busy and idle periods, alternating idle and data-limited periods, or with idle or data-limited periods during slow-start.

C.6. Evaluating TFRC's Response to Idle Periods

In this section we focus on evaluating Revised TFRC's response to idle or data-limited periods.

One drawback to Standard TFRC's strict response to idle or data-limited periods is that it could be seen as encouraging applications to pad their sending rate during idle or data-limited periods, by sending dummy data when there was no other data to send. Because Revised TFRC has a less strict response to data-limited periods than that of Standard TFRC, Revised TFRC also could be seen as giving applications less of an incentive to pad their sending rates during data-limited periods. Work in progress, such as Faster Restart [KFS07], can also decrease an application's incentive to pad its sending rate, by allowing faster start-up after idle periods. Further research would be useful to understand, in more detail, the interaction between TCP or TFRC's congestion control mechanisms, and an application's incentive to pad its sending rate during idle or data-limited periods.

TCP Congestion Window Validation, described in Appendix C.1 above, is an Experimental standard specifying that the TCP sender slowly reduces the congestion window during an idle or data-limited period [RFC2861]. While TFRC and Revised TFRC's responses to idle periods are roughly similar to those of TCP with Congestion Window Validation, Revised TFRC's response to data-limited periods is less conservative than those of TCP with Congestion Window Validation (and Standard TFRC's response to data-limited periods was considerably *more* conservative than those of Congestion Window Validation). Future work could include modifications to this document so that the response of Revised TFRC to a data-limited period includes a slow reduction of the allowed sending rate; Section C specifies a possible mechanism for this. Such a modification would be particularly compelling if Congestion Window Validation became a Proposed Standard in the IETF for TCP.

References

Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3448] Handley, M., Floyd, S., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 3448, January 2003.

Informative References

[BRS99] Balakrishnan, H., Rahul, H., and Seshan, S., "An Integrated Congestion Management Architecture for Internet Hosts," Proc. ACM SIGCOMM, Cambridge, MA, September 1999.

[CCID-4] Floyd, S., and E. Kohler, "Profile for DCCP Congestion Control ID 4: the Small-Packet Variant of TFRC Congestion Control", Work in Progress, February 2008.

[FHPW00] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications", August 2000, Proc SIGCOMM 2000.

[FHPW00a] 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.

[FF99] Floyd, S., and K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet, IEEE/ACM Transactions on Networking, August 1999.

[KFS07] E. Kohler, S. Floyd, and A. Sathiaseelan, "Faster Restart for TCP Friendly Rate Control (TFRC)", Work in Progress, November 2007.

[MAF05] A. Medina, M. Allman, and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet", ACM Computer Communications Review, April 2005.

[PFTK98] 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.

[RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, April 1997.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC2581bis] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", Work in Progress, April 2008.

[RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.

[RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

[RFC3448Err] RFC 3448 Errata,
<http://www.rfc-editor.org/errata_search.php?rfc=3448>.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

[RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, March 2006.

[RFC4828] Floyd, S. and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant", RFC 4828, April 2007.

[W00] Widmer, J., "Equation-Based Congestion Control", Diploma Thesis, University of Mannheim, February 2000,
<http://www.icir.org/tfrc/>.

Authors' Addresses

Sally Floyd
ICSI
1947 Center St, Suite 600
Berkeley, CA 94708
EMail: floyd@icir.org

Mark Handley,
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
UK
EMail: M.Handley@cs.ucl.ac.uk

Jitendra Padhye
Microsoft Research
EMail: padhye@microsoft.com

Joerg Widmer
DoCoMo Euro-Labs
Landsberger Strasse 312
80687 Munich
Germany
EMail: widmer@acm.org