Internet Engineering Task Force
INTERNET-DRAFT
Intended status: Experimental
Expires: January 2009

E. Kohler
UCLA
S. Floyd
ICIR
A. Sathiaseelan
University of Aberdeen

14 July 2008

Faster Restart for TCP Friendly Rate Control (TFRC)
draft-ietf-dccp-tfrc-faster-restart-06.txt

Status of this Memo

Copyright Notice

Abstract

TCP-Friendly Rate Control (TFRC) is a congestion control mechanism for unicast flows operating in a best-effort Internet environment. This document introduces Faster Restart, an optional mechanism for safely improving the behavior of interactive flows that use TFRC. Faster Restart is proposed for use with TFRC and with TFRC-SP, the Small Packet variant of TFRC. We present Faster Restart in general terms as a congestion control mechanism, and further discuss Faster Restart for Datagram Congestion Control Protocol (DCCP) Congestion Control IDs 3 and 4.

Table of Contents

NOTE TO RFC EDITOR: PLEASE DELETE THIS NOTE UPON PUBLICATION.

Changes from draft-ietf-dccp-tfrc-faster-restart-05.txt:

* Updated application-limited behavior for Revised TFRC in Table 1, to reflect changes to rfc3448bis.

* Updated description of code in rfc3448bis to reflect changes in that document.

Changes from draft-ietf-dccp-tfrc-faster-restart-04.txt:

* Changed "RTO" to "NFT".
  Changed the targeted idle period to the configurable DelayTime.
  Feedback from Gerrit Renker.

* Removed Section 4.1 on the receive rate, after it is made into an Errata for RFC 4342. Feedback from Gerrit Renker.

* General editing from Gorry Fairhurst and Arjuna, and additional reporting on simulations.

* Added a section on Interoperability Issues.

* Specified CCID 3 and 4 impact in the introduction.

Changes from draft-ietf-dccp-tfrc-faster-restart-03.txt:

* Deleted ping packets, and the section about the implementation of ping packets in DCCP.

* In Section 3.2, calls to
  "Update X_active_recv and X_fast_max;" and
  "Interpolate X_fast_max;"
  had been reversed accidentally. Put them back in the right order.

* Changed Intended Status back to Experimental (where it started out).

* General editing is response to feedback from Gorry.

* Added simulation tests to the list in the section on simulations:
  (1) simulations
  with a worst-case scenario of high congestion, all flows using TFRC, all flows having various idle times, all flows using Faster Restart, and variable arrival rates for the TFRC flows (to create variable levels of congestion). And compare this to the same scenario with no flows using Faster Restart. (2) scenarios with

transient changes from routing changes and from variable traffic. The goal is to explore worse-case scenarios showing off the worst aspects of Faster Restart.

* Targeted an idle period of at most six minutes, not thirty minutes. Feedback from Gorry and Ian McDonald.

* Added a section of whether Faster Restart encourages flows to pad their sending rate during idle periods.

* Didn't implement suggestion from Lachlan Andrew to decay from quadrupling to doubling the sending rate gradually. The last more-than-doubling of the sending rate is probably not a quadrupling in any case, since the allowed sending rate is not increased due to quadrupling to more than X_fast_max.

Changes from draft-ietf-dccp-tfrc-faster-restart-02.txt:

* Deleted proposed response to dealing with X_recv for idle or data-limited periods; RFC3448bis now deals with this instead.

* Deleted the Receive Rate Length option. Also removed all text about using the inflation factor to reduce X_recv_in based on the sender's idle time.

* Moved TFRC changes and DCCP-specific changes to separate sections.

* Revised draft to refer to RFC3448bis instead of to RFC3448. This included modifying sections on "Feedback Packets" and "Nofeedback Timer".

* Said that CCID 3 could calculate the receive rate only for one RTT, rather than for longer, after an idle period. (When used with RFC3448bis, it shouldn't affect performance one way or another).

Changes from draft-ietf-dccp-tfrc-faster-restart-01.txt:

* Added a sentence to Abstract about DCCP.

* Added some text to the Introduction,

* Added sections on "Minimum Sending Rate", "Send Receive Rate Length Feature", "Nofeedback Timer", and "Simulations of Faster Restart".

* Added an Appendix on "Simulations".

Changes from draft-ietf-dccp-tfrc-faster-restart-00.txt:

* Added mechanisms for dealing with a more general problem with idle periods. This includes a section of "Receive Rate Adjustment".

END OF NOTE TO RFC EDITOR.

1. Introduction

This document defines congestion control mechanisms that improve the performance of occasionally idle flows using TCP-Friendly Rate Control (TFRC) [RFC3448] [RFC3448bis]. A data-limited or idle flow uses less than its allowed sending rate for application-specific reasons, such as lack of data to send. The responses of Standard TFRC [RFC3448], and Revised TFRC [RFC3448bis] to long idle or data-limited periods are summarized in Table 1 below, and the responses of Standard TCP [RFC2581] and TCP with Congestion Window Validation [RFC2861] are described in Appendix C of [RFC3448bis]. All of these mechanisms allow a flow to recover from a long idle period by ramping up to the allowed sending rate or window. This document specifies mechanisms that allow TFRC to start at a higher sending rate after an idle period, and to ramp up faster to the old sending rate after an idle period.

As this draft is being written, Standard TFRC is specified in [RFC3448], and TFRC is in the process of being revised, as Revised TFRC, in [RFC3448bis]. When [RFC3448bis] is approved as a Proposed Standard document, this draft will be revised, with the phrase "Standard TFRC" replaced by "Old TFRC", and other language changes as appropriate.

For Standard TFRC as specified in [RFC3448], a TFRC flow may not send more than twice X_recv, the rate at which data was received at the receiver over the previous RTT. Thus in Standard TFRC the previous receive rate limits the sending rate of applications with highly variable sending rates, forcing the applications to ramp up, by doubling their sending rate each round-trip time, from the earlier data-limited rate to the sending rate allowed by the throughput equation. TFRC's nofeedback timer halves the allowed sending rate after each nofeedback timer interval (at least four round-trip times) in which no feedback is received. One result is that applications must slow-start after being idle for any significant length of time, in the absence of mechanisms such as Quick-Start [RFC4782] and Quick-Start for DCCP [GA08].

For Revised TFRC as specified in [RFC3448bis], the previous receive rate is not used to limit the sending rate during data-limited

periods. Thus, unlike [RFC3448], in [RFC3448bis] applications with highly variable sending rates are not limited by the previous receive rates. However, [RFC3448bis] is like [RFC3448] in that the nofeedback timer is used to halve the allowed sending rate after each nofeedback timer interval in which no feedback is received. With [RFC3448] the allowed sending rate is not reduced below two packets per RTT during idle periods, and with [RFC3448bis] the allowed sending rate is not reduced below the allowed initial sending rate during idle periods.

This behavior is safe, though conservative, for best-effort traffic in the network. A silent application stops receiving feedback about the condition of the current network path, and thus should not be able to send at an arbitrary rate. A data-limited application stops receiving feedback about whether current network conditions would support higher rates. However, this behavior also affects the perceived performance of interactive applications such as voice. Connections for interactive telephony and conference applications, for example, will usually have one party active at a time, with seamless switching between active parties. TFRC's reduction of the allowed sending rate, and slow-starting back to a higher sending rate, after every switch between parties could seriously degrade perceived performance. Some of the strategies suggested for coping with this problem, such as sending padding data during application idle periods, might have worse effects on the network than simply switching onto the desired rate with no slow-start.

There is some justification for somewhat accelerating the slow start process after idle periods, as opposed to at the beginning of a connection. A flow that fairly achieves a sending rate of X has proved, at least, that some path between the endpoints can support that rate. The path might change, due to endpoint reset or routing adjustments; or many new connections might start up, significantly reducing the application's fair rate. However, it seems reasonable to allow an application to possibly contribute to limited transient congestion in times of change, in return for improving application responsiveness.

This document suggests a relatively simple approach to this problem. Standard TFRC [RFC3448] specifies that the allowed sending rate is never reduced below two packets per RTT as the result of a nofeedback timer after an idle period. Following [RFC3390], CCID-3 [RFC4342] and Revised TFRC [RFC3448bis] specify that the allowed sending rate is never reduced below the TCP initial sending rate of two or four packets per RTT, depending on packet size, as the result of a nofeedback timer after an idle period. Faster Restart doubles this allowed sending rate after idle periods. Thus, the sending rate after an idle period is not reduced below a rate Y between four and

eight packets per RTT, depending on the packet size. The rate Y is restricted to at most 8760 bytes per RTT (which is twice TCP's maximum allowed initial window size).

In addition, because flows already have some (possibly old) information about the path, Faster Restart allows flows to quadruple their sending rate in every congestion-free RTT, instead of doubling, upwards towards the previously achieved rate. When the TFRC sender detects congestion, the sender leaves Faster Restart and changes into congestion avoidance. These changes are summarized in the table below. In this document, "NFT" refers to the NoFeedback Timer interval for TFRC; this is roughly equivalent to the Retransmit TimeOut (RTO) interval for TCP.

```
-----------------------------------------------------------------
- Standard TFRC -
-----------------------------------------------------------------
Idle period:
  Halve allowed sending rate each NFT, not below two packets per RTT.
  After sending again, double the sending rate each RTT.
Application-limited period:
  Send at most twice X_recv.
  As a result, at most double the sending rate each RTT.
-----------------------------------------------------------------

-----------------------------------------------------------------
- Revised TFRC -
-----------------------------------------------------------------
Idle period:
  Halve allowed sending rate each NFT, not below initial sending rate.
  After sending again, double the sending rate each RTT.
Application-limited period:
  If no loss, send at most twice max (X_recv_set), including old values
    of X_recv going back to just before the data-limited interval was
    entered.
  If loss, reduce saved values of X_recv.
-----------------------------------------------------------------

-----------------------------------------------------------------
- Revised TFRC with Faster Restart -
-----------------------------------------------------------------
Idle period:
  Halve allowed sending rate each NFT, not below twice initial rate.
    (Specified in Section 3.2.)
  After sending again, quadruple the sending rate towards old rate.
    (Specified in Section 3.1.)
Application-limited period:
  Sending rate not limited by X_recv.
-----------------------------------------------------------------
```

Table 1: Behavior of TFRC, with and without Faster Restart.

The congestion control mechanisms defined here are intended to apply to any implementations of TFRC, including that in DCCP's CCID 3 and CCID 4 [RFC4342], [CCID4]. These mechanisms change only CCID 3 and 4 sender behavior and do not change DCCP packets in externally visible ways (except in that the sending rate will be higher after an idle period). This reduces interoperability concerns. Any DCCP CCID 3 or 4 sender MAY therefore use Faster Restart algorithms at its discretion, without negotiation with the corresponding receiver.

While we also believe that TCP could safely use a similar Faster Restart mechanism, we do not specify it here. Our assumption is that flows that are sensitive to restrictions to the sending rate after idle periods are more likely to use TFRC than to use TCP or TCP-like congestion control.

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].

The Faster Restart mechanism refers to several existing TFRC state variables, including the following:

R: The RTT estimate.

X: The current allowed sending rate in bytes per second.

p: The recent loss event rate.

X_recv:
The rate at which the receiver estimates that data was received since the last feedback report was sent.

s: The packet size in bytes.

Faster Restart uses the following variable from [RFC3448bis]:

recv_limit:
The limit on the sending rate that is computed from the receive rate.

Faster Restart also introduces new state variables to TFRC, as follows:

X_active_recv:
The receiver's estimated receive rate reported during a recent active sending period. An active sending period is a period in which the sender has not experienced a loss event. X_active_recv is initialized to 0 until there has been an active sending period, and X_active_recv is reduced after a loss event.

T_active_recv:
The time at which X_active_recv was measured. T_active_recv is initialized to the start time of the connection.

recover_rate:
The minimum restart rate allowed by Faster Restart after an idle period. Note that Faster Restart flows can drop below this rate as the result of experienced congestion (e.g. actual loss feedback). Recover_rate is defined as follows:

recover_rate = min(8*s, max(4*s, 8760 bytes))/R.

Faster Restart also uses the following, which could be implemented as a temporary variable:

X_fast_max:
The rate at which the sender should stop quadrupling its sending rate, and return to at most doubling its sending rate.

Other variables have values as described in [RFC3448] and [RFC3448bis].

3. Faster Restart: Changes to TFRC

3.1. Feedback Packets

The Faster Restart algorithm replaces the lines in step (4) of Section 4.3, "Sender Behavior When a Feedback Packet is Received", of [RFC3448bis] that specify the limitation on the sending rate calculated from the reported receive rates. [RFC3448bis] allows the sender to slow-start back up to the previous sending rate after an idle period, doubling its sending rate after each round-trip time.

This document specifies a mechanism so that during recovery from an idle period, the TFRC sender can quadruple its sending rate each (congestion-free) round-trip time, until it reaches its old sending rate before the idle or data-limited period. This modification uses three new variables: X_active_recv specifies the maximum receive rate achieved before the idle period, T_active_recv specifies the time of the last update of X_active_recv, and X_fast_max specifies the adjusted rate at which the sender should stop quadrupling its sending rate and continue to its default behavior of doubling its sending rate.

The procedure "Update X_active_recv and X_fast_max" below increases the two variables in response to increases in the reported receive rate and reduces them after a report of a lost packet or an indication of congestion (e.g. an ECN-marked packet).

```
    Update X_active_recv and X_fast_max:
        If (the feedback packet does not indicate a loss or mark,
              and X_recv >= X_fast_max)
            X_active_recv = X_fast_max = X_recv,
            T_active_recv = current time.
        Else if (the feedback packet DOES indicate a loss or mark,
              and X_recv < X_fast_max)
            X_active_recv = X_fast_max = X_recv/2,
            T_active_recv = current time.
```

The parameter X_active_recv gives an upper bound on the rate achievable through Faster Restart, and is only modified by the "Update X_active_rate and X_fast_max" procedure. This modification is based on the contents of the feedback packet and the value of X_fast_max. X_active_recv is updated as the connection achieves higher congestion-free transmit rates. X_active_recv is reduced on congestion feedback, to prevent an inappropriate Faster Restart until a new stable active rate is achieved. Specifically, when congestion feedback is received at a low sending rate, the sender reduces X_active_recv to X_recv/2, allowing a limited Faster Restart up to a likely-safe rate.

For some transport protocols using TFRC, the feedback packets might report the loss event rate, but not explicitly report lost or marked packets. For such protocols, the sender in the "Update X_active_rate and X_fast_max" procedure can infer that a feedback packet indicates a loss or mark by looking at the reported loss event rate. If the current or previous feedback packet reported an increase in the loss event rate, then the current feedback packet is assumed to indicate a loss or mark. (If the previous feedback packet reported an increase in the loss event rate, then a loss event began in the interval covered by that feedback packet. However, the loss event can cover up to a round-trip time of data, so the second half of the loss event, including additional lost or marked packets, could be covered by the second feedback packet.)

The "Interpolate X_fast_max" procedure determines X_fast_max, the adjusted rate at which Faster Restart should stop. The procedure sets X_fast_max to something between zero and X_active_recv, depending on the time since X_active_recv was last updated. The procedure allows full Faster Restart up to the old sending rate X_active_recv after a short idle period, but requires more conservative behavior after a longer idle period. Thus, if at most DecayTime has elapsed since the last update of X_active_recv, for a default DecayTime of two minutes, then X_fast_max is set to X_active_recv. If 3*DecayTime or more has elapsed, X_fast_max is set to zero. Linear interpolation is used between these extremes.

The default DecayTime of two minutes is chosen to strike a balance between the needs of applications, and the time intervals over which connections might reasonably quadruple back up to their old sending rates after idle periods. In terms of the needs of applications, models of voice traffic generally use average idle times between 0.5 and two seconds [JS00] (Section 3). However, in terms of changes in path characteristics, Faster Restart does not assume that the previous sending rate is valid after an idle period; Faster Restart simply assumes that a connection may *quadruple* rather than *double* its sending rate up to the previous rate. Therefore, while an overly long DecayTime is not likely to lead to congestion collapse, it could result in unnecessary packet drops, and therefore in reduced performance for the application itself. Path congestion levels can change over time scales of round-trip times, which are generally between 10 and several hundred milliseconds; more dramatic changes in path characteristics (e.g., routing changes, changes in link bandwidth) happen less frequently. For now, the DecayTime may be a configurable parameter. Future work may shed more light on optimum values for DecayTime.

```
Interpolate X_fast_max:
   // If achieved X_active_recv <= 1 minute ago,
   //    set X_fast_max to X_active_recv;
   // If achieved X_active_recv >= 3 minutes ago,
   //    set X_fast_max to zero;
   // If in between, interpolate.
   delta_T = now - T_active_recv;
   F = (6 min - min(max(delta_T, 2 min), 6 min)) / (2 min);
   X_fast_max = F * X_active_recv;
```

The pseudocode above uses the temporary variables delta_T and F.

Faster Restart replaces the following lines from step (4) of Section 4.3 of [RFC3448bis]:

```
    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);
    }
```

with the following:

```
    Interpolate X_fast_max;
    Update X_active_recv and X_fast_max;
    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);
            If (recv_limit < X_fast_max)
                recv_limit = min (2*recv_limit, X_fast_max);
        }
    } Else {                     // typical behavior
        Update X_recv_set();
        recv_limit = 2 * max (X_recv_set);
        If (recv_limit < X_fast_max)
            recv_limit = min (2*recv_limit, X_fast_max);
    }
```

In summary, when a feedback packet is received, as specified in [RFC3448bis], then the sender updates the round-trip time estimate and the NFT (NoFeedback Timer), and updates X_recv_set, the set of recent X_recv values, and then executes the procedure above. X_fast_max always represents the interpolated value from highest X_recv reported since the last loss event. However, because

X_recv_set contains only X_recv values from the most recent two round-trip times, the calculated recv_limit could be less than X_fast_max. In this case, recv_limit is doubled, up to at most X_fast_max. Faster Restart's doubling of recv_limit allows the TFRC sender to quadruple its sending rate each round-trip time after an idle period.

3.2. Nofeedback Timer

Section 4.4 of [RFC3448bis] specifies when the allowed sending rate is halved after the nofeedback timer expires. In particular, [RFC3448bis] specifies that if the sender has been idle since the nofeedback timer was set, then the allowed sending rate is not reduced below recover_rate, which in [RFC3448bis] is set to the initial_rate of W_init/R, for:

```
W_init = min(4*s, max(2*s, 4380)),
```

for segment size s. In contrast, this document sets recover_rate to twice the initial_rate, as follows:

```
recover_rate = 2*W_init/R;
```

4. Faster Restart Discussion

Standard TCP has historically dealt with idleness and data-limited flows either by keeping cwnd entirely open ("immediate start") or by entering slow-start, as recommended in RFC 2581 in response to an idle period. The first option is too liberal, the second too conservative. Clearly a short idle or data-limited period is not a new connection: the sending rate maintained before the idle or data-limited period shows that previously, the connection could fairly sustain some rate without adversely impacting other flows. However, longer idle periods are more problematic. Idle periods of many minutes would seem to require slow-start.

RFC 2861 [RFC2861] gives a moderate mechanism for TCP, where the congestion window is halved for every retransmit timeout interval that the sender has remained idle, down to the initial window, and the window is re-opened in slow-start when the idle period is over. TFRC in [RFC3448bis] roughly follows [RFC2861] for the response to an idle period. Unlike [RFC2861], however, [RFC3448bis] follows Standard TCP in its responses to a data-limited period, and does not reduce the allowed sending rate in response to data-limited periods.

4.1. Worst-Case Scenarios

Faster Restart should be acceptable for TFRC if its worst-case scenarios are acceptable. Realistic worst-case scenarios might include the following scenarios:

- o Path changes: The path changes and the old rate is not acceptable on the new path. RTTs are shorter on the new path too, so Faster Restart takes bandwidth from other connections for multiple RTTs, not just one. (This can happen with TCP or with TFRC without Faster Restart, but Faster Restart could make this behavior more severe.)

- o Synchronized flows: Several connections enter Faster Restart simultaneously. If the path is congested, the extra load resulting from Faster Restart could be twice as bad as the extra load if the connections had simply slow-started from their allowed initial sending rate.

- o Many forms of burstiness: Variable-rate connections using Faster Restart share the congested link with short TCP or DCCP connections starting and stopping, with initial windows of three or four packets. The aggregate traffic could also include TCP connections with short quiescent periods (e.g., web browsing sessions using HTTP 1.1), or bursty higher-priority traffic. As a result of the bursty traffic, the aggregate arrival rate varies from one RTT to the next. The transient congestion will be particularly severe if the congested link is an access link instead of a backbone link; the level of statistical multiplexing on an access link may not be sufficiently high to "smooth out" the burstiness.

- o Wireless links: The network allocates capacity based on traffic conditions, as in some current wireless technologies, such as Bandwidth on Demand (BoD) links [RFC3819] where capacity is variable and dependent on several parameters other than network congestion. In this case, the old sending rate might not be acceptable after a change in capacity for the wireless link during an idle period.

Further analysis is required to analyze the effects of these scenarios.

4.2. Incentives for applications to send unnecessary packets during idle or data-limited periods

How does Faster Restart affect an application's incentive to pad its sending rate by sending unnecessary packets during idle or data-

limited periods? We would like to limit an application's incentive to pad its sending rate during idle or data-limited periods; if all applications were to pad their sending rates, it could reduce the available bandwidth, and degrade the performance for all flows on the congested link.

With Standard TFRC as specified in [RFC3448], a data-limited TFRC flow may not send more than twice X_recv, the rate at which data was received at the receiver over the previous RTT. Thus, with Standard TFRC, one could argue that a variable-rate application over an uncongested path does have some incentive to pad its sending rate.

With Revised TFRC as specified in [RFC3448bis], the allowed sending rate after an idle period is larger than the allowed sending rate with Standard TFRC. Further, with Revised TFRC the receive rate reported in feedback packets is not used to limit the sending rate during data-limited periods. Thus, with Revised TFRC an application has less incentive to pad its sending rate than with Standard TFRC. However, with Revised TFRC an application could have some incentive to pad its sending rate just enough to maintain the status of "data-limited" instead of "idle", by sending at least one packet every four round-trip times.

By allowing TFRC to revert to its old sending rate more quickly after an idle period, Faster Restart could reduce an application's incentive to pad its sending rate.

4.3. Interoperability Issues

Faster Restart is a sender-side only modification to TFRC, and is intended to work with any TFRC receiver using the same transport protocol. The current standard for TFRC is RFC 3448. After [RFC3448bis] is standardized, the authors of this document will verify that Faster Restart works with either an RFC3448 or an RFC3448bis receiver.

4.3.1. Interoperability Issues with CCID-3 and the RFC 4342 Errata

For the particular case of TFRC as used in CCID-3 or CCID-4 in DCCP, there are currently two variants of CCID-3 receivers. For TFRC as specified in [RFC3448], the receiver reports the receive rate measured over the most recent round-trip time. In contrast, for CCID-3 as specified in [RFC4342], the receiver reports the receive rate measured over the interval since the last feedback packet was received. These two methods can differ for feedback packets sent after a loss event or after an idle period. To correct this, the RFC 4342 Errata [RFC4342Errat] now specifies that the receiver reports the receive rate measured over the most recent round-trip time, as in

RFC 3448.

Because Faster Restart is being specified only for a sender using [RFC3448bis], and not for a sender using [RFC3448], Faster Restart in CCID-3 should interoperate with a CCID-3 receiver as specified in [RFC4342], with a CCID-3 receiver as specified in [RFC4342] and updated by the RFC 4342 Errata, or with a CCID-3 receiver as specified in [RFC4342] updated by both the RFC 4342 Errata and by [RFC3448bis]. In particular, with Faster Restart in CCID-3 (or CCID-4) with RFC3448bis, the sender's sending rate is not limited by the first feedback packet received after an idle period, so Faster Restart should perform well even with a CCID-3 (or CCID-4) receiver following RFC 4342 and not updated by the RFC 4342 Errata.

4.4. Faster Restart for TFRC-SP

We note that Faster Restart with TFRC-SP [RFC4828] is considerably more restrained than Faster Restart with TFRC. In TFRC-SP, the sender is restricted to sending at most one packet every Min Interval.

5. Simulations of Faster Restart

Some test case scenarios based on simulation analysis are described in Appendix A. These simulations follow the guidelines set in [RFC4828]. These are:

1. Fairness to standard TCP and TFRC: The simulation tests examine whether flows that use Faster Restart allow TCP and TFRC flows can achieve their share of the path capacity.

2. Fairness within Faster Restart: The simulation tests examine how multiple competing Faster Restart flows share the available capacity among them.

3. Response to transient events: The simulation tests examine how a Faster Restart flow reacts to a sudden congestion event.

4. Behavior in a range of environments: Tests assess a range of bandwidths, RTTs, and varying idle periods.

A set of initial simulation results will be described in [S08]. We note some of the important results here.

- Faster Restart does improve the performance of a flow after an idle period by faster restarting when compared to TFRC. The results indicate that the worst case packet delay distribution is small for Faster Restart than for TFRC.

- o The effect of Faster Restart restarting after an idle period seems to have an effect on other competing flows only when the Faster Restart flow has a high sending rate before it enters the idle period.

- o When the Faster Restart flows experience losses and hence reduce their rates to a lower rate prior to entering an idle period, the effect of faster restarting is similar to that of slow-start.

A later version of this draft will provide more discussion on these results in the appendix and implications will be noted here.

6. Implementation Issues

TBA

7. Security Considerations

TRFC security considerations are discussed in [RFC3448]. DCCP security considerations are discussed in [RFC4340]. Faster Restart adds no additional security considerations.

8. IANA Considerations

There are no IANA considerations.

9. Thanks

We thank the DCCP Working Group for feedback and discussions; we particularly thank Gorry Fairhurst. We thank Vlad Balan and Gerrit Renker for pointing out problems with the mechanisms discussed in previous versions of the draft.

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, Proposed Standard, January 2003.

[RFC3448bis] Handley, M., Floyd, S., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", internet draft draft-ietf-dccp-rfc3448bis-06.txt, work-in-progress, April 2008.

[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.

Informative References

[CCID4] Floyd, S., and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)", Internet-Draft draft-ietf-dccp-ccid4-02.txt, work in progress, February 2008.

[GA08] "Quick-Start for the Datagram Congestion Control Protocol (DCCP)", Internet-Draft draft-fairhurst-tsvwg-dccp-qs-03.txt, work in progress, June 2008.

[JS00] W. Jiang and H. Schulzrinne, Analysis of On-Off Patterns in VoIP and Their Effect on Voice Traffic Aggregation, Proceedings of the Ninth Conference on Computer Communications and Networks (ICCCN), October 2000.

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

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

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

[RFC3819] Karn, P., Ed., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", RFC 3819, July 2004.

[RFC4342Errat] RFC Errata for RFC 4342, URL "http://www.rfc-editor.org/errata.php".

[RFC4782] Floyd, S., Allman, M., Jain, A., and P. Sarolahti, "Quick-Start for TCP and IP", RFC 4782, June 2006.

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

[S08] Sathiaseelan, A., Faster Restart - Analysis, URL www.erg.abdn.ac.uk/users/arjuna/faster-restart.pdf, to appear.

A. Appendix: Simulations

This appendix describes a set of initial test case scenarios for simulation analysis of Faster Restart. The simulation results use the ns-2 simulator.

Several types of flows are considered:

- o Bulk TCP Flows.
- o Interactive (short) TCP Flows.
- o TFRC Flows with and without Faster Restart.
- o TFRC-SP Flows with and without Faster Restart.

The implications on other flows (e.g. using UDP) may be extrapolated from this.

For these simulations, we consider two application rates.

- o Small media flows: These have a similar rate to voice over IP with a media bit rate of 64 Kbps (using segments of 160 bytes and a nominal transmit rate of 8 KBps).
- o Large media flows: These have a similar rate to medium quality video over IP with a media bit rate of 512 Kbps (using segments of size 1000 bytes and a nominal transmit rate of 64 KBps).

The simulations will model the effect of an idle period in which the application does not attempt to send any data for a period of time, then resumes transmission. Various idle times are considered.

The simulation scenarios include the following. These are intended to be illustrative, rather than exact models of the application behavior.

- o Performance of a long-lived (bulk) TCP flow (e.g. FTP) with TFRC flows (with and without Faster Restart): The test scenario would involve a single large FTP flow with varying number of large media

flows. Each large media flow becomes idle for one second and then restarts. The FTP flow starts during the idle period. The throughput performance of the single FTP flow would be plotted for varying number of large media flows. Does the single FTP flow get at least 1/n share of the bandwidth, where TFRC flows decrease the bandwidth received by the TCP flow?

o Performance of small TCP flows (HTTP) with TFRC flows with and without Faster Restart: The test scenario would involve a single large media flow which runs for ten seconds, is idle in the time interval [2, 3], and then restarts. At three seconds, a number of HTTP flows are started. The min, max and median of the request/response time of these HTTP flows would be plotted. Do the request/response times of these HTTP flows differ? If so, by how much?

o High-congestion test: In a worst-case scenario with high congestion, all flows use TFRC, with a range of arrival times and idle times. The simulations are run both with and without Faster Restart. How does the use of Faster Restart affect the aggregate packet drop rate?

o Transient changes: The first worst-case scenario with transient changes includes a routing change, where the new path has less bandwidth than the old path. The second scenario with transient changes includes transient congestion from a sudden increase in traffic. This increase in traffic could be from long-lived TCP traffic, or from higher-priority traffic, or from many new TFRC sessions. The transient congestion could be particularly severe if the congested link is an access link instead of a backbone link. The third scenario with transient changes could include a wireless link with variable bandwidth, as discussed earlier in Section 4. A fourth scenario would involve a mobility event that results in an increase in the round-trip time. In all cases, the simulations are run both with and without Faster Restart. How does the use of Faster Restart affect the aggregate packet drop rate?

o An ideal scenario showing the benefits of Faster Restart: A scenario with an uncongested network, just a few TFRC flows, comparing the per-packet delay distribution with and without Faster Restart. Without Faster Restart, there should be a few packets in each flow with very large delay times, from waiting at the sender until they can be sent.

o A scenario showing the benefits (to the flow, not to competing traffic) of padding during idle periods: Are there any scenarios where Faster Restart *increases* a flow's incentives to pad its

sending rate during idle or under-utilized periods?

Authors' Addresses

Eddie Kohler
4531C Boelter Hall
UCLA Computer Science Department
Los Angeles, CA 90095
USA

Email: kohler@cs.ucla.edu

Sally Floyd
ICSI Center for Internet Research
1947 Center Street, Suite 600
Berkeley, CA 94704
USA

Email: floyd@icir.org

Arjuna Sathiaseelan
Electronics Research Group
University of Aberdeen
Aberdeen
UK

Email: arjuna@erg.abdn.ac.uk

Full Copyright Statement

Intellectual Property