```
Internet Engineering Task Force                              E. Kohler
INTERNET-DRAFT                                                    UCLA
draft-ietf-dccp-tfrc-faster-restart-02.ps                      S.Floyd
Expires: September 2007                                           ICIR
Intended status: Proposed Standard                     A. Sathiaseelan
                                                University of Aberdeen
                                                          2 March 2007
```

Faster Restart for TCP Friendly Rate Control (TFRC)

Status of this Memo

Abstract

TCP-Friendly Rate Control (TFRC) is a congestion control mechanism for unicast flows operating in a best-effort Internet environment [RFC3448]. 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 both the default TFRC and with the small packet variant of TFRC [TFRCSP]. We present Faster Restart in general terms as a congestion control mechanism, and further describe how to implement Faster Restart in Datagram Congestion Control Protocol (DCCP) Congestion Control IDs 3 and 4 [RFC4342], [CCID4].

Table of Contents

1. Introduction

This document defines congestion control mechanisms that improve the performance of data-limited and/or occasionally idle TCP-Friendly Rate Control (TFRC) [RFC3448] flows. A data-limited and/or idle flow uses less than its fair share of path bandwidth for application-specific reasons, such as lack of data to send. Existing TFRC (and TCP) mechanisms prevent such a flow from quickly ramping up to its fair share of path bandwidth. We present mechanisms that allow applications to ramp up faster, in a controlled way.

In any RTT, a TFRC flow may not send more than twice X_recv, the amount that was received in the previous RTT. The TFRC nofeedback timer reduces this number by half during each nofeedback timer interval (at least four RTT) in which no feedback is received. The effect of this is that applications must slow start after going idle for any significant length of time, in the absence of mechanisms such as Quick-Start [RFC4782]. Similarly, X_recv forces applications with variable sending rates that wish to ramp up from an application-limited rate up to a fair-share rate to do so using slow start.

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 slowly-sending application stops receiving feedback about whether current network conditions would support higher rates. But this behavior can damage 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. A slow start on every switch between parties may 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 or slow 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 contribute to transient congestion in times of change, in return for improving application responsiveness.

This document suggests a relatively simple approach to this problem. Some protocols using TFRC [RFC4342] already 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 an idle or slow period. [RFC3390]. Faster Restart doubles this allowed sending rate after idle periods: that the allowed sending rate is never reduced below four packets per RTT, or eight packets per RTT for small packets, as the result of an idle or slow period. 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, up to the previously achieved rate. Any congestion event stops this faster restart and switches TFRC into congestion avoidance.

This document also addresses a more general problem with idle periods. The first feedback packet sent after an idle period may report an artificially low X_recv, since the time interval used by the receiver to calculate X_recv may include the idle period as well as active periods on either side. This low value will artificially depress the sender’s send rate. DCCP’s TFRC

CCIDs 3 and 4 [RFC4342], [CCID4] report X_recv using a Receive Rate option. We suggest a change to this Receive Rate option that lets the sender detect and compensate for such problems.

The congestion control mechanisms here are intended to apply to any implementations of TFRC, including that in DCCP's CCID 3 and CCID 4 [RFC4342], [CCID4]. While we also believe that TCP could safely use similar mechanisms, we do not specify them here.

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].

3. Faster Restart Congestion Control

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

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.

X_calc
: The safe rate determined by the TCP throughput equation. Calculated from p, R, and s.

Faster Restart also introduces two 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 was neither idle nor in faster restart. It is initialized to 0 until there has been an active sending period.

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

X_active_min_rate
: The minimum restart rate allowed by Faster Restart in the presence of idle and/or data-limited periods. Note that Faster Restart flows can drop below this rate as the result of actual loss feedback. X_active_min_rate is defined as follows:

```
X_active_min_rate := min(8*s, max(4*s, 8760 bytes)).
```

Other variables have values as described in [RFC3448].

3.1. Minimum Sending Rate

The TFRC specification allows a TFRC endpoint to go completely silent when the sending application runs out of data to send. When Faster Restart is used, however, the transport layer MUST send a minimum of X_ping/s packets per second, where X_ping is defined as
X_ping = min(X, s/4R).

That is, the transport layer will send at least one packet per four round-trip times, as allowed by the current allowed sending rate X. These packets give the endpoint a continuing stream of RTT samples and information about network congestion. Extra packets generated by the transport layer to maintain a minimum sending rate SHOULD NOT be reported to the receiving application.

DCCP implementations MUST use DCCP-Data or DCCP-DataAck packets with a zero-length application data area for packets sent to maintain a minimum sending rate. To that end, this document modifies RFC 4340's behavior with respect to zero-length application data area DCCP-Data and DCCP-DataAck packets. RFC 4340, Section 5.4, specifies that:

> A DCCP-Data or DCCP-DataAck packet may have a zero-length application data area, which indicates that the application sent a zero-length datagram. This differs from DCCP-Request and DCCP-Response packets, where an empty application data area indicates the absence of application data (not the presence of zero-length application data). The API SHOULD report any received zero-length datagrams to the receiving application.

This document revises this statement as follows.

> A DCCP-Data or DCCP-DataAck packet may have a zero-length application data area. Such packets may be sent by congestion control algorithms to maintain a minimum sending rate. As in DCCP-Request and DCCP-Response packets, an empty application data area indicates the absence of application data. The API MUST NOT report any received zero-length datagrams to the receiving application. The API SHOULD report an error when a sending application attempts to send a zero-length datagram.

3.2. Receive Rate Adjustment

The X_recv values reported by a TFRC receiver may be artificially depressed by idle periods. The sender can properly detect and account for such X_recv values, given some information about whether a reported X_recv includes information about an idle period. We describe the relevant algorithm in the context of an implementation in DCCP's CCID 3 and 4. This implementation adds a new option to required feedback packets, namely Receive Rate Length.

```
+--------+--------+--------+--------+--------+
|11000100|00000101|   Receive Rate Length    |
+--------+--------+--------+--------+--------+
 Type=196   Len=5
```

Receive Rate Length (24 bits)

> The Receive Rate Length reports the number of packets used to calculate the Receive Rate, minus one. If a feedback packet's Receive Rate was calculated using data packet sequence numbers S1...S2, inclusive, where S2 is the feedback packet's Acknowledgement Number, then Receive Rate Length will be set to S2 − S1. Thus, a Receive Rate Length of zero indicates that one packet was used to calculate Receive Rate.

The Receive Rate Length option allows senders to adjust Receive Rates before using them in TFRC calculations. The first adjustment applies to any Receive Rate options, with or without Receive Rate Lengths.

- Assume that the sender receives two feedback packets with Acknowledgement Numbers A1 and A2, respectively. Further assume that the sender sent no data packets in between Sequence Numbers A1+1 and A2. (All those packets must have been pure acknowledgements, Sync and SyncAck packets, and so forth.) Then the sender MAY, at its discretion, ignore the second feedback packet's Receive Rate option. Note that when the sender decides to ignore such an option, it MUST NOT reset the nofeedback timer as it normally would; the nofeedback timer will go off as if the second feedback packet had never been received.

The second adjustment applies only to packets containing a Receive Rate Length as well as a Receive Rate. If a packet contains a Receive Rate option but not a Receive Rate Length, then the sender MUST use that Receive Rate as is. We refer to the original Receive Rate, as encoded in the option, as X_recv_in.

- Assume that the sender receives a feedback packet with Acknowledgement Number S2 and Receive Rate Length RRL. Let S1 = S2 − RRL; then the feedback packet's Receive Rate was calculated using sequence numbers S1...S2, inclusive. Assume that the sender sent packet S1 at time T1, and packet S2 at time T2. If T1 = T2, then X_recv_in MUST be used as is. Otherwise, assume that in that interval, the sender was idle for a total of I seconds. Here, "idle" means that the sender had nothing to send for a contiguous period of at least one-half round trip time. (Note that this definition of idleness is less conservative than that applied to the Faster Restart algorithm.) Then the sender MAY act as if the feedback packet specified a Receive Rate of

  X_recv_in*(T2 − T1 + I)/(T2 − T1),

  rather than the nominal Receive Rate of X_recv_in. The inflation factor, (T2 − T1 + I)/(T2 − T1), compensates for the idle periods by removing their effect.

3.2.1. Send Receive Rate Length Feature

The Send Receive Rate Length feature lets DCCP CCID 3 and 4 endpoints negotiate whether the receiver MUST provide Receive Rate Length options on its feedback packets. DCCP A sends a "Change R(Send Receive Rate Length, 1)" option to ask DCCP B to send Receive Rate Length options as part of its acknowledgement traffic.

Receive Rate Length has feature number 196 and is server-priority. It takes one-byte Boolean values. DCCP B MUST send Receive Rate Length options on its feedback packets when Send Receive Rate Length/B is one, although it MAY send Receive Rate Length options even when Send Receive Rate Length/B is zero. Values of two or more are reserved. A CCID 3 half-connection starts with Send Receive Rate Length equal to zero.

3.3. Feedback Packets

The Faster Restart algorithm replaces for the 4th step of Section 4.3, "Sender behavior when a feedback packet is received", of [RFC3448]. The replacement code has two goals:

1. It keeps track of the active receive rate, X_active_recv. This parameter models the connection's highest recent loss- and mark-free fair transmit rate, and represents an upper bound on the rate achievable through faster restart. Thus, X_active_recv is increased as the connection achieves higher congestion-free transmit rates, and reduced on congestion feedback, to prevent inappropriate Faster Restart until a new stable active rate is achieved. Specifically, on congestion feedback at low rates, the sender sets X_active_recv to X_recv/2; this allows limited Faster Restart up to a likely-safe rate, and

lowers the likelihood that badly-timed transient congestion will wholly cripple the Faster Restart mechanism.

2. It adjusts the receive rate, X_recv, more aggressively during faster restart periods, up to the limit of X_active_recv.

The code works in four phases. The first phase adjusts the feedback packet's X_recv to make sure it does not drop too low as the result of a slow send rate.

The second phase determines X_fast_max, the adjusted rate at which Faster Restart should stop. Full Faster Restart up to X_active_recv should be allowed for short idle periods, but more conservative behavior should prevail after longer idle periods. Thus, if 10 minutes or less have elapsed since the last active-period measurement (T_active_recv), the code sets X_fast_max to the full value of X_active_recv. If 30 minutes or more have elapsed, X_fast_max is set to 0. Linear interpolation is used between these extremes.

The second phase adjusts X_active_recv based on the feedback packet's contents and the value of X_fast_max.

Finally, the third phase sets X based on X_fast_max, X_recv, and X_calc, the calculated send rate. Several temporary variables are used, namely X_fast_max, delta_T, F, and X_recv_limit.

```
To update X when you receive a feedback packet
----------------------------------------------
/* First phase.  Adjust X_recv so send rate doesn't drop
   below X_active_min_rate as the result of an idle and/or
   slow period. */
If the feedback packet does not indicate a loss or mark
      and the old X_recv >= X_active_min_rate/2, then
   X_recv := max(X_recv, X_active_min_rate/2).

/* Second phase.  Calculate X_fast_max */
/* If achieved X_active_recv <= 10 minutes ago, end
   Faster Restart at the full last fair rate; if achieved
   X_active_recv >= 30 minutes ago, don't do Faster Restart;
   in between, interpolate. */
delta_T := now - T_active_recv,
F := (30 min - min(max(delta_T, 10 min), 30 min)) / 20 min,
X_fast_max := F*X_active_recv.

/* Third phase.  Update X_active_recv */
If the feedback packet does not indicate a loss or mark
      and X_recv >= X_fast_max, then
   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, then
   X_active_recv := X_fast_max := X_recv/2,
   T_active_recv := current time.

/* Fourth phase.  Calculate X */
X_recv_limit := 2*X_recv.
If X_recv_limit < X_fast_max,
   X_recv_limit := min(4*X_recv, X_fast_max).
If p > 0,
   Calculate X_calc using the TCP throughput equation.
   X := max(min(X_calc, X_recv_limit), s/t_mbi).
Else
   If (t_now - tld >= R)
      X := max(min(2*X, X_recv_limit), s/R);
      tld := now.
```

3.4. Nofeedback Timer

RFC 3448, Section 4.4, specifies that the sending rate is cut in half when the TFRC nofeedback timer expires. This is accomplished by reducing X_recv. Faster Restart changes this algorithm so that the sending rate never drops below 4 packets per RTT, or 8 packets per RTT for small packets, as the result of an idle period. In particular, Step 1) of the algorithm executed as a result of a nofeedback timer is changed to the following:

```
If the sender has sent no data whatsoever since the
      time the nofeedback timer was set,
      and X_active_min_rate/2 <= X_recv <= X_active_min_rate,
   X_recv := X_active_min_rate/2.
Else if X_calc > 2*X_recv, then
   X_recv := max(X_recv/2, s/(2*t_mbi)).
Else
   X_recv := X_calc/4.
```

4. Faster Restart Discussion

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. The first option is too liberal, the second too conservative. Clearly a short idle or data-limited period is not a new connection: recent evidence shows that the connection could fairly sustain some rate. However, longer idle periods are more problematic, and idle periods of many minutes would seem to require slow start. RFC 2861 [RFC2861], which is fairly widely implemented [MAF04], gives a moderate mechanism for TCP, where the congestion window is halved for every round-trip time that the sender has remained idle, and the window is re-opened in slow-start when the idle period is over.

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

- The path changes and the old rate isn’t acceptable on the new path. RTTs are shorter on the new path too, so Faster Restart clobbers other connections for multiple RTTs, not just one.
- Two (or more) connections enter Faster Restart simultaneously. The packet drop rate can be twice as bad, for one RTT, than if they had slow-started after their idle periods.
- In addition to connections Fast-Restarting, there are short TCP or DCCP connections starting and stopping all the time, with initial windows of three or four packets. There are also TCP connections with short quiescent periods (web browsing sessions using HTTP 1.1). The audio and video connections have idle periods. The available bandwidth might vary over time because of bandwidth used by higher-priority traffic. All of this might happen at once, so the aggregate arrival rate naturally varies from one RTT to the next. And the congested link is an access link, not a backbone link, so the level of statistical multiplexing may not be sufficiently high for connections to obtain a deterministic estimate of the fair rate.
- The network allocates capacity based on traffic conditions, as happens 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.

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

We note that Faster Restart in TFRC-SP [TFRCSP] is considerably more restrained that Faster Restart in the default TFRC. In TFRC-SP, the sender is restricted to sending at most one packet every Min Interval. Similarly, Faster Restart in the default TFRC is more restrained than Faster Restart would be if added to TCP; TFRC is controlled by a sending rate, while TCP is controlled by a window, and could send in a very bursty pattern without rate-based pacing.

5. Simulations of Faster Restart

Some test case scenarios based on simulation analysis are described in Appendix A. These simulation follow the guidelines set in [TFRCSP]. 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 its fair share rate of the path capacity.
2. Fairness within FR: The simulation tests examine how multiple competing FR flows share the available capacity among them.
3. Response to transient events: The simulation tests examine how a FR flow reacts to a sudden congestion event.
4. Behaviour in a range of environments: Tests assess a range of bandwidth, RTTs, and varying idle periods.

>>> 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

DCCP security considerations are discussed in [RFC4340]. Faster Restart adds no additional security considerations. XXX WE WILL PROBABLY BE REQUIRED TO ADD SOME STUFF HERE

8. IANA Considerations

This document allocates two values in the "Profile for DCCP Congestion Control ID 3: TFRC Congestion Control Parameters" registry. Specifically, it allocates Option Type 196 for the Receive Rate Length option, and Feature Number 196 for the Send Receive Rate Length feature.

9. Thanks

We thank the DCCP Working Group for feedback and discussions, including Gorry Fairhurst. We especially thank Vlad Balan 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.

[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-floyd-dccp-ccid4-00.txt, work in progress, October 2006.

[MAF04] Medina, A., Allman, M., and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet", May 2004, URL "http://www.icir.org/tbit/".

[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.

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

[TFRCSP] Floyd, S., and E. Kohler, "TCP Friendly Rate Control (TFRC): the Small-Packet (SP) Variant", Internet-Draft draft-ietf-dccp-tfrc-voip-07.txt, work in progress, November 2006.

A. Appendix: Simulations

This appendix describes a set of initial test case scenarios for simulation analysis of Faster Restart. The topology will be the classic dumb-bell topology used in many simulations of TCP.

Six types of flow are considered:

- Bulk TCP Flows.
- Interactive (short) TCP Flows.
- TFRC Flows.
- TFRC Flows that employ FR.
- TFRC-SP Flows.
- TFRC Flows that employ FR (TFRC-SP).

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

For these simulations, we consider three application-limited rates.

- The first resembles constant bit rate (CBR) voice over IP with a media bit rate of 64 kbps (using packets of size 160 bytes and a nominal transmit rate of 8000Bps).
- The second resembles constant bit rate (CBR) medium quality video over IP with a media bit rate of 512 kbps (using packets of size 1000 bytes and a nominal transmit rate of 64000Bps).

- The third class uses an unspecified upper limit on the sending rate, but experiences period of idleness.

These are intended to be illustrative, rather than exact models of the application behaviour.

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.

In the first case, we shall examine periods of idleness of 1s, 10s, and 30s with a path RTT of 50ms, 300ms.

The scenarios to be examined are:

- Performance of a long-lived (bulk) TCP flow (e.g. FTP) with TFRC (with and without FR): The test scenario would involve a single large FTP flow with varying number of CBR flows. Each CBR flow becomes idle for 10s 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 CBR flows. Simulations would be performed by varying parameters such as CBR rate and number of silence periods. Does the single FTP flow get at least 1/n share of the bandwidth, where ’n’ is the number of TFRC flows and the single TCP flow? Does the single TCP flow get less share of the bandwidth while competing with FR flows when compared to TFRC flows?
- Fairness test: The test scenario would involved ’n’ number CBR and long lived TCP flows. The CBR flows become idle for 10s and then restarts. During the silence period, the FTP flows arrive. Do all flows get atleast 1/n share of the bandwidth? Jain’s Fairness Index [JCH84] would be an appropriate measure.
- Performance of small TCP flows (HTTP) with TFRC with and without FR: The test scenario would involve a single CBR flow running for 50s, becomes ilde between 20s and 30s and then restarts. At 30.s, a number of HTTP flows are started. The min, max and median of the request/response time of these HTTP flows would be plotted. Simulations would be performed by varying several parameters such as CBR rate, bottleneck bandwidth, delay and queue size. Do the request/response times of these HTTP flows differ? If so, by how much?

Authors’ Addresses

Eddie Kohler <kohler@cs.ucla.edu>
4531C Boelter Hall
UCLA Computer Science Department
Los Angeles, CA 90095
USA

Sally Floyd <floyd@icir.org>
ICSI Center for Internet Research
1947 Center Street, Suite 600
Berkeley, CA 94704
USA

Arjuna Sathiaseelan <arjuna@erg.abdn.ac.uk>
Electronics Research Group
University of Aberdeen
Aberdeen
UK