Internet Engineering Task Force
INTERNET-DRAFT
Intended status: Proposed Standard
Expires: January 2008

E. Kohler
UCLA
S. Floyd
ICIR
A. Sathiaseelan
University of Aberdeen
8 July 2007

Faster Restart for TCP Friendly Rate Control (TFRC)
draft-ietf-dccp-tfrc-faster-restart-03.ps

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 both the default 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 describe how to implement Faster Restart in 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-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 data-limited or 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 TCP

[RFC2581], TCP with Congestion Window Validation [RFC2861], Standard TFRC [RFC3448], and Revised TFRC [RFC3448bis] in response to long idle or data-limited periods 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 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 or data-limited period.

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 limitation from the 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 application-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 going idle for any significant length of time, in the absence of mechanisms such as Quick-Start [RFC4782].

For Revised TFRC as specified in [RFC3448bis], during data-limited periods, the receive rate reported in feedback packets is not used to limit the sending rate. Thus, unlike [RFC3448], in [RFC3448bis] applications with highly variable sending rates are not limited by the receive rates from data-limited periods. Like [RFC3448], in [RFC3448bis] the nofeedback timer is used to halve the allowed sending rate after each nofeedback timer interval in which no feedback is received, though with [RFC3448bis] an exception is made for idle periods, when the allowed sending rate is not reduced below the allowed initial sending rate.

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. However, 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 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. Following [RFC3390], some protocols using TFRC [RFC4342] [RFC3448bis] 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 a nofeedback timer after an idle or as a result of receive rate report during a slow period. Faster Restart doubles the 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.

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 towards the previously achieved rate. Any congestion event stops this faster restart and switches TFRC into congestion avoidance.

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 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 or data-limited periods are more likely to use TFRC that 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 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 was neither idle nor in faster restart. X_active_recv is initialized to 0 until there has been an active sending period.

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

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

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

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

3. Faster Restart: Changes to TFRC

3.1. Minimum Sending Rate

TFRC 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. However, losses of these packets MUST be used to update the allowed sending rate.

3.2. Feedback Packets

The Faster Restart algorithm replaces the line

recv_limit = 2 * max (X_recv_set);

in step (4) of Section 4.3, "Sender Behavior When a Feedback Packet is Received", of [RFC3448bis]. This line specifies the limitation on the sending rate from the recent receive rate, and in [RFC3448bis] allows the sender to slow-start back up after an idle or data-limited period, doubling its sending rate after each round-trip time.

This document replaces the line above, so that during recovery from an idle period, the TFRC sender can quadruple its sending rate, instead of just doubling it, up towards its old sending rate before the idle period. This modification uses three new variables, X_active_recv specifying the maximum receive rate achieved before the idle period, T_active_recv specifying the time of the last update of X_active_recv, and X_fast_max specifying the adjusted rate at which the sender should stop quadrupling its sending rate, and return to at most 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 following a lost or 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 feedbacak 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, on congestion feedback at low rates, 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 explicity 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 10 minutes or less have elapsed since the last update of X_active_recv, then X_fast_max is set to X_active_recv. If 30 minutes or more have elapsed, X_fast_max is set to zero. Linear interpolation is used between these extremes.

```
Interpolate X_fast_max:
   // If achieved X_active_recv <= 10 minutes ago,
   //    set X_fast_max to X_active_recv;
   // If achieved X_active_recv >= 30 minutes ago,
   //    set X_fast_max to zero;
   // If 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;
```

This procedure uses the temporary variables delta_T and F.

The entire procedure replaces the following line from step (4) of Section 4.2 of [RFC3448bis]:

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

with the following:

```
Update X_active_recv and X_fast_max;
Interpolate X_fast_max;
recv_limit :=  2 * max (X_recv_set);
If (recv_limit < X_fast_max)
   recv_limit := min(2*recv_limit, X_fast_max);
```

This allows the TFRC sender to quadruple its sending rate during Faster Restart. We note that the variable X_fast_max can be implemented as a temporary variable.

3.3. 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: DCCP-specific Specifications

4.1. DCCP: Implementation of Minimum Sending Rate

Section 3.1 above specifies that when TFRC uses Faster Restart, the sender must send occasional ping packets during idle times. This section specifies the implementation of these ping packets for [RFC4342] and [CCID4].

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 usual packet receiving API MUST NOT report any received zero-length datagrams to the receiving application. For instance, when a receiving application asks the API to return the next received packet, the API should always return a packet with at least one byte of application data. (However, a special-purpose API, such as an API designed to report connection

liveness, MAY report received zero-length datagrams.)

4.2. DCCP: Receive Rate Adjustment

Unlike [RFC3448] and [RFC3448bis], Section 8.3 of DCCP's [RFC4342] specifies that the Receive Rate option reports the receive rate since the last feedback packet was sent. In contrast, Section 6.2 of [RFC3448] and of [RFC3448bis] specify that the feedback packet reports the receive rate over the last round-trip time. As a result, the receive rate reported by [RFC4342] differs from that of TFRC for a feedback packet after an idle period; the receive rate of TFRC reports the receive rate over the entire idle period. The receive rate reported by [RFC4342] also differs from that of TFRC for an early feedback packet reporting a new loss event. In this document, we specify that [RFC4342] and [CCID4] should use the definition of the receive rate as specified in [RFC3448] and [RFC3448bis].

In particular, the fourth paragraph in Section 6 of [RFC4342] is changed from:

> 2. A Receive Rate option, defined in Section 8.3, specifying the
>   rate at which data was received since the last DCCP-Ack was
>   sent.

to:

> 2. A Receive Rate option, defined in Section 8.3, specifying the
>   rate at which data was received over the last round-trip time.

Similarly, the first paragraph in Section 8.3 of [RFC4342] is changed from:

> This option MUST be sent by the data receiver on all required acknowledgements. Its four data bytes indicate the rate at which the receiver has received data since it last sent an acknowledgement, in bytes per second. To calculate this receive rate, the receiver sets t to the larger of the estimated round-trip time and the time since the last Receive Rate option was sent. (Received data packets' window counters can be used to produce a suitable RTT estimate, as described in Section 8.1.) The receive rate then equals the number of data bytes received in the most recent t seconds, divided by t.

to:

> This option MUST be sent by the data receiver on all required acknowledgements. Its four data bytes indicate the rate at which the receiver has received data over the last round-trip time, in bytes per second. To calculate the time interval t for calculating this receive rate, the receiver follows Section 6.2 of [RFC3448bis], or roughly equivalently, Section 6.2 of [RFC3448]. (Received data packets' window counters can be used to produce a suitable RTT estimate, as described in Section 8.1.) The receive rate then equals the number of data bytes received in the most recent t seconds, divided by t.

A feedback packet sent in response to the first packet received after an idle period reports a receive rate of one packet per round-trip time. As a change from [RFC3448], [RFC3448bis] doesn't use the receive rate reported in such packets to reduce the allowed sending rate. Because [RFC3448bis] doesn't use the receive rate to reduce the allowed sending rate when the data sender was data-limited over the entire interval covered by the receive rate, the

DCCP sender that follows [RFC3448bis] generally would not use the receive rate from an interval that did not include data packets.

To be precise, we specify language for DCCP so that if the entire period covered by the last feedback packet doesn't include any data packets, then the sender doesn't use the reported receive rate to reduce the sending rate, even if the sender was not data-limited over than interval. To do that, we add the following:

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, inclusive. (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.

4.3. DCCP: The Receive Rate Length

[The Receive Rate Length option in earlier versions of this document has been deleted. The Receive Rate Length option is not needed for feedback packets sent after an idle period, because of changes in [RFC3448bis]. The Receive Rate Length option should not be used for the sender to account for short idle periods within a feedback period. The Receive Rate Length option is also not needed for the case discussed above when the sender is not data-limited, but the data sending rate is less than one packet per round-trip time, and the interval covered by the feedback packet doesn't include any data packets; this case is dealt with above without the use of the Receive Rate Length.

5. 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: recent evidence shows that 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.

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

- Path changes: 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 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.]

- 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.
- Many forms of burstiness: 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 could 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. The transient congestion could 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.
- 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.

We note that Faster Restart in TFRC-SP [RFC4828] is considerably more restrained than 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.

6. 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 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.

7. Implementation Issues

TBA

8. 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

9. IANA Considerations

There are no IANA considerations.

10. 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.

[RFC3448bis] Handley, M., Floyd, S., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", internet draft draft-ietf-dccp-rfc3448bis-02.txt, work-in-progress, July 2007.

[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-ccid4-01.txt, work in progress, June 2007.

[JCH84] R.K. Jain, Dah-Ming Chiu, and Willian R. Hawe, A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems, DEC Technical tleport TR-301, Digital Equipment Corporation, September 1984.

[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.

[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.

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