DCCP Media Strategies
Internet Draft **T. Phelan**
Document: draft-phelan-dccp-media-00.txt **Sonus Networks**
Expires: May 2004 November 2003

Strategies for Media Applications Using DCCP

Abstract

This document discusses strategies for using DCCP as the transport protocol for streaming media applications. Of particular interest is how media streams can be adapted to the smoothly varying transmit rate requirements of CCID3, or TCP-Friendly Rate Control (TFRC). Also explored is the resulting network behavior of streams using these strategies and the fairness between these streams and TCP-based streams.

Table of Contents

1. Introduction

The Datagram Congestion Control Protocol (DCCP), as defined in [DCCP], is a transport protocol that implements a congestion-controlled unreliable service. Currently, there are two congestion control algorithms for DCCP, referred to by Congestion Control Identifiers (CCIDs). CCID2 is a TCP-like additive increase multiplicative decrease algorithm [CCID2]. CCID3 is an implementation of TCP-Friendly Rate Control (TFRC) [RFC 3448]. The congestion control algorithm in effect for each direction of data transfer is chosen at connection setup time.

The IAB has expressed concerns over the continuing use of UDP without congestion control for voice and other media traffic [IABCONG]. One of the motivating factors for the development of DCCP was to provide a protocol with congestion control that media applications could use to handle these issues.

This document explores strategies for streaming media applications to use when employing DCCP with CCID3. In addition, this document looks at the resulting network performance from the points of view of both the media applications and TCP-based applications sharing the same network resources. The document explores the issue of the fairness of DCCP-based media applications to TCP-based applications, similar to [IABCONG], but also explores the fairness of TCP-based applications to DCCP-based media applications.

The approach here is one of successive refinement. Strategies are described and their strengths and weaknesses are explored. New strategies are then presented that improve on the previous ones and the process iterates. The intent is to illuminate the issues, rather than to jump to solutions, in order to provide guidance to application designers.

This document is meant to be a complement to, and at a higher level than, the DCCP User Guide [DCCPUG].

1.1 Types of Media Applications

While all streaming media applications have some characteristics in common (e.g. data must arrive at the receiver at some minimum rate for reasonable operation), other characteristics (e.g. tolerance of delay) vary considerably from application to application. For the purposes of this document, it seems useful to divide streaming media applications into three subtypes:

- o One-way pre-recorded media
- o One-way live media
- o Two-way interactive media

The relevant difference, as far as this discussion goes, between recorded and live media is that recorded media can be transmitted as fast as the network allows (assuming adequate buffering at the receiver) -- it could be viewed as a special file transfer operation. Live media can't be transmitted faster than the rate that it's encoded.

The difference between one-way and two-way media is the sensitivity to delay. For one-way applications, delays from transmit at the sender to playout at the receiver of several or even tens of seconds are acceptable. For two-way applications delays from transmit to playout of as little as 150 to 200 ms are often problematic.

1.2 Stream Switching

The discussion here assumes that media transmitters are able to provide their data in a number of encodings with various bit rate requirements, as described in [SWITCH], and are able to dynamically change between these encodings with low overhead. It also assumes that switching back and forth between coding rates does not cause user annoyance.

Given the current state of codec art, these are big assumptions. The algorithms and results described here, however, hold even if the media sources can only supply media at one rate. Obviously the statements about switching encoding rates don't apply, and an application with only one encoding rate behaves as if it is simultaneously at its minimum and maximum rate.

For convenience in the discussion below, assume that all media streams have two encodings, a high bit rate and a low bit rate, unless otherwise indicated.

1.3 Media Buffers

Many of the strategies below make use of the concept of a media buffer. A media buffer is a first-in-first-out queue of media data. The buffer is filled by some source of data and drained by some other sink. This provides rate and jitter compensation between the source and the sink.

Media buffer contents are measured in seconds of media play time, not bytes or packets. Media buffers are completely application-level constructs and are separate from transport-layer transmit and receive queues.

1.4 TFRC Basics

The job of mapping media applications onto the packet formats and connection handshake mechanisms of DCCP proper is straightforward, and won't be dealt with here. The problem for this document is how media stream applications can make use of and adapt to the idiosyncrasies of TCP-Friendly Rate Control (TFRC), as implemented in CCID3.

Data streams controlled by TFRC must vary their transmission rates in ways that, at first blush, seem at odds with common media stream transmission practices. Some particular considerations are:

- o Slow Start -- A connection starts out with a transmission rate of one packet per round trip time (RTT). After the first RTT, the rate is doubled each RTT until a packet loss event is seen. At this point the transmission rate is halved and we enter the next phase of operation. It's likely that in many situations the initial transmit rate is slower than the lowest bit rate encoding of the media. This will require the application to deal with a ramp up period.

- o Capacity Probing -- If the application transmits for some time at the maximum rate that TFRC will allow, TFRC will continuously raise the allowed rate until a packet loss event is encountered. This means that if an application wants to transmit at the maximum possible rate, packet loss will not be an exceptional event, but will happen routinely in the course of probing for more capacity.

- o Idleness Penalty -- TFRC follows a "use it or lose it" policy. If the transmitter goes idle for a few RTTs, as it would if, for instance, silence suppression were being used, the transmit rate returns to two packets per RTT, and the application must then slowly ramp up to higher rates. This makes silence suppression problematic.

- o Contentment Penalty -- TFRC likes to satisfy greed. If you are transmitting at the maximum allowed rate, TFRC will try to raise that rate. However, if your application has been transmitting below the maximum allowed rate, the maximum allowed rate will not be increased, no matter how long it has been since the last increase. This can create problems when attempting to shift to a higher rate encoding.

- o Packet Rate, not Bit Rate -- TFRC controls the rate that packets may enter the network, not bytes. To respond to a lowered transmit rate you must reduce the packet transmission rate. Making the packets smaller while still keeping the same packet rate will not be effective.

o Smooth Variance of Transmit Rate -- The strength and purpose of TFRC (over TCP-Like Congestion Control, CCID2) is that it smoothly decreases the transmission rate in response to recent packet loss events, and smoothly increases the rate in the absence of loss events. This smoothness is at odds with most media stream encodings, where the transition from one rate to another is usually a step function.

2. First Attempt -- One-way Pre-recorded Media

The first strategy takes advantage of the fact that the data for pre-recorded media can be transferred to the receiver as fast as the network will allow it, assuming that the receiver has sufficient buffer space.

2.1 Strategy 1

Assume a recorded program resides on a media server, and the server and its clients are capable of stream switching between two encoding rates, as described in section 1.2.

The client (receiver) implements a media buffer as a playout buffer. This buffer is potentially big enough to hold the entire recording. The playout buffer has three thresholds: a low threshold, a playback start threshold, and a high threshold, in order of increasing size. These values will typically be in the several to tens of seconds range. The buffer is filled by data arriving from the network, and drained at the decoding rate necessary to display the data to the user. Figure 1 shows this schematically.

```
                          high threshold
                              |  playback start threshold
                              |    |  low threshold
+-------+                     |    |    |
| Media |  transmit at    +---v----v----v--+
| File  |---------------->| Playout buffer |-------> display
|       |  TFRC max rate  +----------------+ drain at
+-------+                 fill at network    decode rate
                          arrival rate
```

Figure 1: One-way pre-recorded media.

During the connection the server needs to be able to determine the depth of data in the playout buffer. This could be provided by direct feedback from the client to the server, or the server could estimate its depth (e.g. the server knows how much data has been sent, and how much time has passed).

To start the connection, the server begins transmitting data in the high bit rate encoding as fast as TFRC allows. Since TFRC is in slow start, this is probably too slow initially, but eventually the rate should increase to fast enough and more. As the client receives data from the network it adds it to the playout buffer. Once the buffer depth reaches the playback start threshold, the receiver begins draining the buffer and playing the contents to the user.

If the network has sufficient capacity, TFRC will eventually raise the transmit rate to greater than necessary to keep up with the decoding rate, the playout buffer will back up as necessary, and the entire program will eventually be transferred.

If the TFRC transmit rate never gets fast enough, or a loss event makes TFRC drop the rate, the receiver will drain the playout buffer faster than it is filled. If the playout buffer drops below the low threshold the server switches to the low bit rate encoding. Assuming that the network has a bit more capacity than the low bit rate requires, the playout buffer will begin filling again.

When the buffer crosses the high threshold the server switches back to the high encoding rate. Assuming that the network still doesn't have enough capacity for the high bit rate, the playout buffer will start draining again. When it reaches the low threshold the server switches again to the low bit rate encoding. The server will oscillate back and forth like this until the connection is concluded.

If the network has insufficient capacity to support the low bit rate encoding, the playout buffer will eventually drain completely, and playback will need to be paused until the buffer refills to some level (presumably the playback start level).

Note that, in this scheme, the server doesn't need to explicitly know the rate that TFRC has determined; it simply always sends as fast as TFRC allows (perhaps alternately reading a chunk of data from disk and then blocking on the socket write call until it's transmitted). TFRC shapes the stream to the network's requirements, and the playout buffer feedback allows the server to shape the stream to the application's requirements.

2.2 Issues With Strategy 1

The advantage of this strategy is that it provides insurance against an unpredictable future. Since there's no guarantee that a currently supported transmit rate will continue to be supported, the strategy takes what the network is willing to give when it's willing to give it. The data is transferred from the server to the client perhaps faster than is strictly necessary, but once it's there no network problems (or new sources of traffic) can affect the display.

Silence suppression can be used with this strategy, since the transmitter doesn't actually go idle during the silence.

One obvious disadvantage, if the client is a "thin" device, is the large buffer at the client. A subtler disadvantage involves the way TFRC probes the network to determine its capacity. Basically, TFRC does not have an a priori idea of what the network capacity is; it simply gradually increases the transmit rate until packets are lost, then backs down. After a period of time with no losses, the rate is gradually increased again until more packets are lost. Over the long term, the transmit rate will oscillate up and down, with packet loss events occurring at the rate peaks.

This means that packet loss will likely be routine with this strategy. For any given transfer, the number of lost packets is likely to be small, but non-zero. Whether this causes noticeable quality problems depends on the characteristics of the particular codec in use. Adding some redundant or error-correcting data to the stream could perhaps mitigate this effect, although the increase in the amount of data transferred needs to be balanced against the small amount of data that will normally be lost.

On the other hand, since end-to-end delay isn't much of an issue here, another solution could be to use TCP [STD0007] (or SCTP [RFC 2960]) as the transport protocol, instead of DCCP. TCP will vary its rate downward more sharply than TFRC, but it will retransmit the lost packets, and only the lost packets. This will cause slight glitches in the transfer rate surrounding loss events, but in many instances the server will be able to catch back up as the transmit rate increases above the minimum necessary.

3. Second Try -- One-way Live Media

With one-way live media you can only transmit the data as fast as it's created, but end-to-end delays of several or tens of seconds are usually acceptable.

3.1 Strategy 2

Assume that we have a playout media buffer at the receiver and a transmit media buffer at the sender. The transmit buffer is filled at the encoding rate and drained at the TFRC transmit rate. The playout buffer is filled at the network arrival rate and drained at the decoding rate. The playout buffer has a playback start threshold and the transmit buffer has a switch encoding threshold and a discard data threshold. These thresholds are on the order of several to tens of seconds. Switch encoding is less than discard data, which is less than playback start. Figure 2 shows this schematically.

```
                discard data
                  |  switch encoding
                  |   |                 playback start
                  |   |                   |
media   +---------v---v---+          +----v-----------+
------->| Transmit buffer |--------->| Playout buffer |---> display
source  +-----------------+ transmit +----------------+
        fill at             at TFRC rate          drain at
        encode rate                               decode rate
```

Figure 2: One-way live media.

At the start of the connection, the sender places data into the transmit buffer at the high encoding rate. The buffer is drained at the TFRC transmit rate, which at this point is in slow-start and is probably slower than the encoding rate. This will cause a backup in the transmit buffer. Eventually TFRC will slow-start to a rate slightly above the rate necessary to sustain the encoding rate (assuming the network has sufficient capacity). When this happens the transmit buffer will drain and we'll reach a steady state condition where the transmit buffer is empty and we're transmitting at a rate that is probably below the maximum allowed by TFRC.

Meanwhile at the receiver, the playout buffer is filling, and when it reaches the playback start threshold playback will start. After TFRC slow-start is complete and the transmit buffer is drained, this buffer will reach a steady state where packets are arriving from the network at the encoding rate (ignoring jitter) and being drained at the (equal) decoding rate. The depth of the buffer will be the playback start threshold plus the maximum depth of the transmit buffer during slow start.

Now assume that network congestion (packet losses) forces TFRC to drop its rate to below that needed by the high encoding rate. The transmit buffer will begin to fill and the playout buffer will begin to drain. When the transmit buffer reaches the switch encoding threshold, the sender switches to the low encoding rate, and converts all of the data in the transmit buffer to low rate encoding.

Assuming that the network can support the new, lower, rate (and a little more) the transmit buffer will begin to drain and the playout buffer will begin to fill. Eventually the transmit buffer will empty and the playout buffer will be back to its steady state level.

At this point (or perhaps after a slight delay) the sender can switch back to the higher rate encoding. If the new rate can't be sustained the transmit buffer will fill again, and the playout buffer will drain. When the transmit buffer reaches the switch encoding threshold the sender goes back to the lower encoding rate. This

oscillation continues until the stream ends or the network is able to support the high encoding rate for the long term.

If the network can't support the low encoding rate, the transmit buffer will continue to fill (and the playout buffer will continue to drain). When the transmit buffer reaches the discard data threshold, the sender must discard data from the transmit buffer for every data added. Preferably, the discard should happen from the head of the transmit buffer, as these are the stalest data, but the application could make other choices (e.g. discard the earliest silence in the buffer). This discard behavior continues until the transmit buffer falls below the switch encoding threshold. If the playout buffer ever drains completely, the receiver should fill the output with suitable material (e.g. silence or stillness).

Note that this strategy is also suitable for one-way pre-recorded media, as long as the transmit buffer is only filled at the encoding rate, not at the disk read rate.

3.2 Issues with Strategy 2

Silence suppression can be a problem with strategy 2. If the encoding rate is low enough -- if it's in the range used by most telephony applications -- the ramp up to the required rate can be short compared to the buffering, and silence suppression can be used. If the encoding rate is in the range of high-quality music or video, then silence suppression is likely to cause problems.

4. One More Time -- Two-way Interactive Media

Two-way interactive media is characterized by its low tolerance for end-to-end delay, usually requiring less than 200 ms. Rate adapting buffers will insert too much delay so another strategy is needed.

4.1 Strategy 3

To start, the calling party sends an INVITE (loosely using SIP [RFC 3261] terminology) indicating the IP address and DCCP port to use for media at its end. Without informing the called user, the called system responds to the INVITE by connecting to the media port. Both end systems then begin exchanging test data, at the (slowly increasing) rate allowed by TFRC. The purpose of this test data is to see what rate the connection can be ramped up to. If a minimum acceptable rate cannot be achieved within some time period, the call is cleared (conceptually, the calling party hears "fast busy" and the called user is never informed of the incoming call). Note that once the rate has ramped up sufficiently for the highest rate codec there's no need to go further.

If an acceptable rate can be achieved (in both directions), the called user is informed of the incoming call. The test data is continued during this period. Once the called user accepts the call, the test data is replaced by real data at the same rate.

If congestion is encountered during the call, TFRC will reduce its allowed sending rate. When that rate falls below the codec currently in use, the sender switches to a lower rate codec. If the TFRC rate continues to fall past the lowest rate codec, the sender must discard packets to conform to that rate.

If the network capacity is sufficient to support one of the lower rate codecs, eventually the congestion will clear and TFRC will slowly increase the allowed transmit rate. Since the application will continue to transmit at its current codec rate, TFRC will limit this increase to at most twice the current sending rate. If the TFRC rate increases sufficiently for the next codec step, the sender may switch to the higher rate. To avoid ramp-up problems, the high rate codec should be less than twice as fast as the low rate codec. If the network can't support the new rate, eventually congestion will reappear and the sender will fall back to the lower rate. Over time the sender will oscillate between the lower and higher rate with a period equal to the time it takes TFRC to probe between the two rates.

Note that the receiver would normally implement a short playout buffer (with playback start on the order of 100 ms) to smooth out jitter in the packet arrival gaps.

4.2 Issues with Strategy 3

An obvious issue with strategy 3 is the post-dial call connection delay imposed by the slow-start ramp up. This is perhaps less of an issue for two-way video applications, where post-dial delays of several seconds are accepted practice. For telephony applications, however, post-dial delays significantly greater than a second are a problem, given that users have been conditioned to that behavior by the public telephone network.

Strategy 3 is unlikely to support silence suppression well. During the silence period, TFRC will lower the transmit rate to two packets per RTT. After the silence the application will need to ramp up to the necessary data sending rate, perhaps causing some lost data.

There are some telephony codecs and network situations where two packets per RTT are more than the necessary data rate. An application that knows it's in this situation could conceivably use silence suppression, knowing that there's no ramp up needed when it returns to transmission.

The next section explores a more subtle issue.

4.3 A Thought Experiment

In [IABCONG], the authors describe a VoIP demonstration given at the IEPREP working group meeting at the 2002 Atlanta IETF. A call was made from a nearby hotel room to a system in Nairobi, Kenya. The data traveled over a wide variety of interfaces, the slowest of which was a 128 kbps link between an ISP in Kenya and several of its subscribers. The media data was contained in typical RTP/UDP framing, and, as is the usual current practice, the transmitter transmitted at a constant rate with no feedback for or adjustment to loss events. The focus of [IABCONG] was on the fairness of this behavior with regard to TCP applications sharing that Kenyan link.

Let's imagine this situation if we replace the RTP/UDP media application with an RTP/DCCP application using strategy 3. Imagine the media application has two encoding rates that it can switch between at little cost, a high-rate at 32 kbps and a low-rate at 18 kbps (these are bits-on-the-wire per second). Furthermore, at the low rate, the receiver can withstand packet loss down to 14 kbps before the output is unintelligible. These numbers are chosen more for computational convenience than to represent real codecs, but they should be conceptually representative.

Let's also imagine that there is a TCP-based application, say large file transfer, whose connection lifetime is of the same order of magnitude as a voice call.

Now imagine that one media connection and one TCP connection are sharing our Kenyan link. Ideally, the media connection would receive 32 kbps and the TCP application would get the remaining 96 kbps.

The situation is not quite that simple, though. A significant difference between the two applications is their degree of contentment or greediness. If the media application can achieve 32 kbps throughput, it's satisfied, and won't push for more. The TCP application, on the hand, is never satisfied with its current throughput and will always push for more.

Periodically, TCP will probe for more throughput, causing congestion on our link, and eventually lost packets. If some of the media packets are lost, DCCP (through TFRC) will back off its transmit rate, causing the media application to fall back to its low bit rate. Basically, any time our DCCP media application is sharing a chokepoint link with a long-lived TCP application, it is going to be periodically driven to its lowest encoding rate.

This is good behavior when TCP is competing with other greedy applications, but it hardly seems fair to the restrained media application. The TCP application was getting 75% of the link; why couldn't it be content to let the media application have 25%?

4.4 Strategy 3 bis

It seems that a modification to strategy 3 is in order. To counter the greediness of TCP, the media application must get greedy also. It should pad its transmitted data out to the TFRC maximum transmit rate. Actually, it's probably sufficient to pad out only to twice the required rate. It's also probably wise to pad with redundant data, so that packet losses can perhaps be recovered, rather than just using filler.

So, during the test data probe period, the application should continue ramping up to at most twice the high encoding rate. When the call is connected, the application switches over to real data, padded out to the maximum rate achieved during slow-start.

TFRC will respond to packet loss events by gradually lowering its transmit rate. At first the application will be able to reduce the padding, without changing the encoding rate. Once the TFRC rate falls below the high encoding rate the application switches to the low encoding rate, but continues to pad this out to the maximum TFRC rate.

As time passes with no more loss events, TFRC will increase the allowed rate. The application should fill that rate with padding until the high encoding rate is reached, then switch to the high rate.

This removes the restriction that encoding rate steps up must be less than a factor of two. When the TFRC rate drops below the high rate encoding, the application switches to the low rate encoding, but continues to pad out to the TFRC rate. As the TFRC rate rises the application continues to add padding, until the rate has risen sufficiently for the higher rate again.

4.5 Back to the Thought Experiment

Given this modification, our two applications will split the available bandwidth pretty much 50/50. Occasionally, TCP will drive the link into congestion, and TFRC might have to back off, but the reduced rate will just cut down the amount of pad data transmitted. Eventually the applications will jostle each other back to the 50/50 split. In this situation, it should never be necessary for the media application to drop to its low rate encoding.

Now let's consider what happens when a second media application connection is added to the existing situation. During the new connection's test data phase, it will encounter congestion very

quickly, but, after a period of time, it should muscle its way in and the three connections will roughly split the link bandwidth, approximately 42 kbps apiece. We'll assume that this jostling period is within the bounds of the acceptable post-dial delay, and the new connection is admitted.

When we add one more media connection we'll end up with approximately 32 kbps per connection. With the three TFRC and one TCP connection all jostling with each other, some of the media streams will momentarily drop below 32 kbps and need to switch to the low encoding rate.

Adding a fourth media connection will leave approximately 25 kbps per connection, forcing the media connections to all permanently switch to the low encoding rate.

By the time we have six media connections (plus the one TCP connection) we have reduced the per-connection bandwidth share to just over 18 kbps. At this point some media connections are discarding packets as the connections jostle for bandwidth and some TFRC rates drop below 18 kbps.

If we try to introduce another media stream connection, reducing the per-connection share further to 16 kbps, the new connection won't be able to achieve a sufficient rate during the test period, and the connection will be blocked. After a moment of packet loss in the existing connections (during the probe period), things will return back to the 18 kbps per-connection state. We won't be able to add a new media connection until one of the existing connections terminates.

But nothing prevents new TCP connections from being added. By the time we have three more TCP connections (for a total of six media connections and four TCP connections) per-connection share has reduced to just under 13 kbps, and the media applications are unusable. The TCP applications, although slow, are likely still useable, and will continue a graceful decline as more TCP connections are added.

4.6 Fairness

The model used above for the interactions of several TCP and TFRC streams -- roughly equal sharing of the available capacity -- is of course a highly simplified version of the real world interactions. A more detailed discussion is presented in [EQCC], however, it seems that the model used here is adequate for the purpose.

The behavior described above seems to be eminently fair to TCP applications -- a TCP connection gets pretty much the same bandwidth

over a congested link that it would get if there were only other TCP connections.

The behavior also seems fair to the network. It avoids persistent packet loss in the network, as occurs in the behavior model in [IABCONG], by discarding media data at the transmitter.

Just how fair this is to media applications is debatable, but it seems better than the method of feeding packets into the network regardless of the congestion situation, but terminate if not enough packets are delivered, as described in [IABCONG]. A media application can choose to not start a connection, if at the moment there are insufficient network resources. A media connection that encounters major congestion after starting up can choose to wait out the congestion, rather than terminate, since the excess packets are discarded before entering the network. The application can perhaps improve quality in a congested situation by discarding packets intelligently, rather than allowing the network to discard randomly. What the likelihood of a user hanging on through this situation is depends on the length and severity of the incident.

5. Security Considerations

This document discusses strategies media application developers can use to deal with the variations in transmit rate that will arise when DCCP with CCID3 is used as the transport layer. The security of media applications using DCCP is outside of that scope.

6. IANA Considerations

There are no IANA actions required for this document.

7. Informative References

[DCCP] E. Kohler, M. Handley, S. Floyd, J. Padhye, Datagram Congestion Control Protocol (DCCP), June 2003, draft-ietf-dccp-spec-04.txt, work in progress.

[CCID2] S. Floyd, E. Kohler, Profile for DCCP Congestion Control 2: TCP-Like Congestion Control, June 20003, draft-ietf-dccp-ccid2-03.txt, work in progress.

[CCID3] S. Floyd, E. Kohler, J. Padhye, Profile for DCCP Congestion Control 3: TFRC Congestion Control, draft-ietf-dccp-ccid3-03.txt, work in progress.

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

[IABCONG] S. Floyd, J, Kempf, IAB Concerns Regarding Congestion for Voice Traffic in the Internet, October 2003, draft-iab-congestion-01.txt, work in progress.

[SWITCH] P. Gentric, RTSP Stream Switching, February 2003, draft-gentric-mmusic-stream-switching-00.txt, work in progress.

[DCCPUG] D. Lanphear, Datagram Congestion Control Protocol (DCCP) User Guide, October 2002, draft-ietf-dccp-user-guide-00.txt.

[STD0007] Transmission Control Protocol, September 1981, STD 0007, RFC 0793.

[RFC 2960] R. Stewart, et al, Stream Control Transmission Protocol, October 2000, RFC 2960.

[RFC 3261] J. Rosenberg, et al, SIP: Session Initiation Protocol, June 2002, RFC 3261

[EQCC] S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-Based Congestion Control For Unicast Applications: the Extended Version, March 2000, International Computer Science Institute, http://www.icir.org/tfrc/tcp-friendly.TR.pdf

8. Author's Address

Tom Phelan
Sonus Networks
5 Carlisle Rd.
Westford, MA 01886
Phone: 978-589-84546
Email: tphelan@sonusnet.com