DCCP User Guide
Internet Draft T. Phelan
Document: draft-ietf-dccp-user-guide-02~~1~~.txt Sonus Networks
Expires: ~~A~~January~~ugust~~ 2005~~4~~ ~~February~~ July 2004

Datagram Congestion Control Protocol (DCCP) User Guide

Status of this Memo

Abstract

This document is an informative reference discussing strategies for using DCCP as the transport protocol for various applications. The focus is on how applications can make use of the capabilities, and deal with the idiosyncrasies, of DCCP. Of particular interest is how UDP applications, which have traditionally ignored congestion control issues, can adapt to a congestion controlled transport.

Table of Contents

Editor's note: Versions of this draft with marked changes from –01 are available:
Inserts and deletes: http://www.phelan-4.com/dccp/draft-ietf-dccp-user-guide-02-all-diffs.pdf
Inserts only: http://www.phelan-4.com/dccp/draft-ietf-dccp-user-guide-02-inserts.pdf

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 (AIMD) algorithm [CCID2]. CCID3 is an implementation of TCP-Friendly Rate Control (TFRC) [RFC 3448], [CCID3]. The congestion control algorithm in effect for each direction of data transfer is chosen at connection setup time.

Many applications that currently use UDP [RFC 768] are candidates for DCCP. The main difference applications will see between UDP and DCCP is congestion control. Because it is complicated to get right, many UDP applications ignore or greatly simplify congestion control issues, even though this can lead to application and network misbehavior. Because of this, some UDP applications employ strategies that are unsuitable for a congestion-controlled transport. Adapting these applications to DCCP will likely require some new modes of thought.

This document explores issues to consider and strategies to employ when adapting or creating applications to use DCCP. 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.

The reader is assumed to be familiar with the mechanisms of DCCP and the CCIDs.

1.1 Candidate Applications

Many applications that currently use UDP, and some that use TCP, are candidates for DCCP. Basically, applications with some of the following characteristics should consider DCCP:

- o There are flows of packets from one end system to another that are larger than a few handfuls of packets.
- o Lost packets should be ignored or replaced with updated data -- timeliness is preferred over reliability.
- o There is a preference for immediate delivery of packets as they arrive over strict in-order delivery by waiting for out-of-order packets to arrive.
- o There is a preference for immediate transmission of small chunks of data.
- o The large transmission rate variations that are typical of TCP congestion control are problematic.

Major examples of applications that fit these characteristics are streaming media, including Internet telephony, and multiplayer on-line games. Without DCCP, these applications either use UDP and mostly ignore or greatly simplify congestion control issues, or use TCP and live with the timeliness and rate variation problems.

Another major use of UDP involves one-shot request-response cycles, with packet flows limited to at most a few packets in each direction (for example, DNS, SNMP, MGCP). These applications are unlikely to benefit from DCCP.

1.2 Which CCID?

CCID2, TCP-like congestion control, uses a packet-oriented modification of TCP's SACK-based Additive-Increase-Multiplicative-Decrease (AIMD) congestion control [RFC 3517]. CCID2 uses a congestion window (the maximum number of packets in flight) to limit the transmitter. The congestion window is increased by one for each acknowledged packet, or for each window of acknowledged packets, depending on the phase of operation. If any packet is dropped (or ECN-marked; for simplicity in the rest of the document assume that "dropped" equals "dropped or ECN-marked"), the congestion window is halved. This produces a characteristic saw-tooth wave variation in throughput, where the throughput increases linearly up to the network capacity and then drops abruptly (roughly shown in Figure 1~~Figure 1~~).

```
           |
           |      /|    /|    /|    /|    /
           |     / |   / |   / |   / |   /
 Throughput|    /  |  /  |  /  |  /  |  /
           |   /   | /   | /   | /   | /
           |  /    |/    |/    |/    |/
           |
            ----------------------------------
                           Time
```

Figure 1: Characteristic throughput for TCP-like congestion control.

With CCID3, TCP-Friendly Rate Control (TFRC), the immediate response to packet drops is less dramatic. To compensate for this CCID3 is less aggressive in probing for new capacity after a loss. The characteristic throughput graph for a CCID3 connection looks like a flattened sine wave (extremely roughly shown in Figure 2~~Figure 2~~).

```
                 |
                 |    --        --        --
                 |   /  \      /  \      /  \
      Throughput |  /    \    /    \    /    \
                 | /      \  /      \  /      \
                 |-        --        --        -
                 |
                  ----------------------------------
                               Time
```

Figure 2: Characteristic throughput for TFRC congestion control.

The CCID that is appropriate for a given application depends on the tradeoff between the application's sensitivity to abrupt rate changes, and its need to rapidly consume available capacity. Applications that simply want to transfer as much data in the shortest time possible probably should use CCID2. Applications that have some natural limits on transmission rates are probably better served by CCID3. Applications that perform some progressive display of incoming data, and want to avoid abrupt variations in the update rate, would prefer CCID3.

1.3 Document Organization

Sections 2 and 3 explore the specific issues involved in using DCCP for streaming media and interactive game applications, respectively. Many of the issues discussed in these sections are also applicable to other applications. Section 4 discusses capabilities of DCCP not mentioned in the application-specific sections that can be used to offload features that are normally built at the application layer for UDP-based applications. Section 5 discusses the security-related features of DCCP.

2. Streaming Media Applications

The canonical streaming media application emits fixed-sized (often small) RTP/UDP packets at a regular interval [RFC 3550]. It relies on the network to deliver the packets to the receiver in roughly the same regular interval. Often, the transmitter operates in a fire-and-forget mode, receiving no indications of packet delivery. This often still holds true even if RTCP [RFC 3550] is used to get receiver information; it's rare that the RTCP reports trigger changes in the transmitted stream.

The IAB has expressed concerns over the stability of the Internet if these applications become too popular with regard to TCP-based applications [IABCONG]. They suggest that media applications should monitor their packet loss rate, and abort if they exceed certain thresholds. Unfortunately, up until this threshold is reached, the network, the media applications and the other applications are experiencing considerable duress.

DCCP offers an opportunity for media applications to satisfy the IAB concerns in a way that is better for both the network and the applications themselves. However, this does require some rather significant shifts from current practices. These shifts are explored in this section.

2.1 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's useful to divide streaming media applications into three subtypes:

- One-way pre-recorded media
- One-way live media
- 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 [XTIME].

2.2 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 excessive 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.

2.3 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 sink. It 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.

2.4 CCID Choice

For two-way media applications, CCID3 (TFRC) is by far the most appropriate congestion control algorithm. Since CCID2 halves the transmit rate when packets are lost, the media encoding steps would need to be at least a factor of two apart. If the encoding steps are less than a factor of two, the application will need to additive increase up after a packet loss; two-way media applications will rarely be able to afford the delays necessary. With the smooth variations in CCID3 the application has much more freedom to choose encoding rate steps.

One-way applications could possibly use CCID2, since they can tolerate more delay during the rate shifts. However, one-way applications often have self-imposed limits on maximum transmission rates that mean they will be unable to reap the higher throughput benefits of CCID2.

2.5 Variable Rate Media Streams

The canonical media codec encodes its media as a constant rate bit stream. As the technology has progressed from its time-division multiplexing roots, this constant rate stream has become not so constant. Voice codecs often employ silence suppression, where the stream (in at least one direction) goes totally idle for sometimes several seconds while one side listens to what the other side has to say. When the one side wants to start talking again, the codec resumes sending immediately at its "constant" rate.

Video codecs similarly employ what could be called "stillness" suppression. Often these codecs effectively transmit the changes from one video frame to another. When there is little change from frame to frame (~~like~~ as when the background is constant and a talking head is just moving its lips) the amount of information to send is small. When there is a major motion, or change of scene, much more information must be sent. For some codecs, the variation from the minimum rate to the maximum rate can be a factor of ten. Unlike voice codecs, though, video codecs typically never go completely idle.

These abrupt jumps in transmission rate are problematic for any congestion control algorithm. A basic tenet of all existing algorithms assumes that increases in transmission rate must be gradual and smooth to avoid damaging other connections in the network.

CCID3 uses a maximum rate of ~~eight~~ two packets per RTT after an idle period. This rate ~~is likely to~~might support immediate restart of voice data after a silence period, at least when the~~for~~ RTT~~s that are~~is in the suitable range for two-way media. More problematic are the factor of ten variations in video codecs. In some circumstances, CCID3 (and CCID2) allows an application to double its transmit rate over one RTT (assuming no recent packet loss events), but an immediate ten times increase is not possible.

2.6 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 section 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:

- Slow Start -- A connection starts out with a transmission rate of up to four packets per round trip time (RTT). After the first RTT, the rate is doubled each RTT until a packet is lost. At this point the transmission rate is halved and we enter the additive increase 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 and Lost Packets -- If the application transmits for some time at the maximum rate that TFRC will allow without packet loss, TFRC will continuously raise the allowed rate until a packet is lost. This means that, in many circumstances, 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 ~~eight~~ two packets per RTT, and then ~~quadruples~~ doubles every RTT until the previous rate is achieved. This can make restarting after a silence suppression interval 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 is transmitting below the maximum allowed rate, the maximum allowed rate will not be increased higher than twice the current transmit rate, no matter how long it has been since the last increase. This can create problems when attempting to shift to a higher rate encoding, or with video codecs that vary the transmission rate with the amount of movement in the image.

- 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 somewhat at odds with most media stream encodings, where the transition from one rate to another is often a step function.

2.7 First Attempt -- One-way Pre-recorded Media

The first strategy is suitable for use with pre-recorded media, and 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.

2.7.1 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 2.2.

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 3~~Figure 3~~ 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 3: 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.

2.7.2 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 -- it just gets further ahead.

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.

If the DCCP connection uses the Ack Vector option for the DCCP-Ack packets in the return direction, DCCP will be able to tell which packets are lost. An API could inform the application of lost packets, and they could be retransmitted at the application layer. See section 3.2 for more on this.

~~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.~~

2.8 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.

2.8.1 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 4~~Figure 4~~ 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 4: 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 normally 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.

2.8.2 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~~or nonexistent, and silence suppression can be used. If the encoding rate is in the range of high-quality music or video, then silence or stillness suppression ~~is likely~~could~~ to~~ cause problems, although the playout buffer might provide sufficient elasticity to overcome the rate ramping issues.

2.9 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., including jitter buffering at the receiver. Rate adapting buffers will insert too much delay and the slow start period is likely to be noticeable, so another strategy is needed.

2.9.1 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 calling party 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, but should pad its transmission out to the allowed TFRC rate. 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. The application should increase its transmission padding to keep up with the increasing TFRC rate. The application switches back to the higher rate codec when the TFRC rate reaches a sufficient value.

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.

2.9.2 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. On the other hand, the four packets per RTT initial transmit rate is likely to be sufficient for many telephony applications, and the ramp up will be very quick.

Strategy 3 might not support silence suppression well. During the silence period, TFRC will lower the transmit rate to ~~eight~~ 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~~.

However, ~~T~~there are many telephony codecs and network situations where ~~eight~~ two packets per RTT ~~are more than the necessary~~is a sufficient 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 some more subtle issues.

2.9.3 A Thought Experiment

In [IABCONG], the authors describe a VoIP demonstration given at the IEPREP working group meeting at the 2002 Atlanta IETF. A VoIP 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 (although forward error correction was used). 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 is using the Kenyan link. This connection will slow-start up to 32 kbps and then self-limit at that rate. The TFRC maximum rate will continue to increase up to 64 kbps, and then hold because of the lack of demand from the application.

Now add one TCP connection to the 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. Since that rate is twice what the application actually needs, and TFRC makes gradual rate changes, it will be a while before the rate is reduced to below 32 kbps. It's likely that the TCP connection will experience packet loss before that, though, and halve its rate, relieving the congestion.

The media connection is therefore somewhat resilient to the TCP probing. In the steady state, the media connection will transmit at a constant 32 kbps (with occasional lost packets), while the TCP connection will vary between 48 and 96 kbps.

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 coexist, with the media applications receiving 32 kpbs apiece and the TCP connection getting between 32 and 64 kbps. 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 media connection and between 16 and 32 kbps for the TCP 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. During that time it's possible for the TCP application to get more than 32 kbps, but eventually things will even out again.

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 (with nominally 7 kbps of padding).

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 discard 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.

2.9.4 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 the same (or more) 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 is of a user hanging on through this situation depends on the length and severity of the incident.

3. Interactive Games

Another application area that could benefit from the use of DCCP is real-time interactive, multi-player, on-line games. These games usually consist of a set of clients (players) that display the game environment to and take commands from a user, and a server computer that maintains the entire game state. Massively Multi-Player (MMP) games can use a grid of server computers connected in a peer-to-peer fashion to distribute the game state computation and increase the number of simultaneous players into the hundreds of thousands or even millions [MMPGRID].

Communications between the various components (client-to-server, server-to-client and server-to-server) usually take two forms. Transient data would like to be delivered as soon as possible, but if lost will be replaced by later data. For example, a player moving through the game space will send a continuous stream of "move-to" messages. If one is lost it isn't worth retransmitting -- the next "move-to" message will give better information. On the other hand, actions that represent permanent changes in the game state must be reliably delivered (e.g., "you're-dead" messages).

It's important for the receiver to able to determine the sequence of the received data, although ~~delivery of data received out-of-order shouldn't wait for reordering~~data received out of order should be delivered immediately, without waiting for the missing data. If two "move-to" messages are reordered it's important to know which one was sent first for the object to end up in the right location, but if the two moves are delivered out of order the late one can be discarded.

Messages are often given an application-layer header, which usually includes sequence numbers and reliability requirements. Each message is usually encapsulated in a single UDP packet and transient information is transmitted in fire-and-forget mode. Persistent data requires an acknowledgement from the receiver, and a retransmit timer at the sender.

The devices would like to transmit their state changes as frequently as possible, as this leads to smoother rendering of the changes at the user display. However, a device can often generate data faster than the network or receiver can handle it, so some rate limits must be applied. Many applications handle this by simply setting a peak data rate limit, rather than dynamically responding to network conditions.

If DCCP were used instead of UDP, a multi-player game application could offload much of work required for the functions of rate limiting, partial reliability and sequencing.

3.1 Rate Limiting

The main purpose of DCCP is to provide congestion control for unreliable streams -- to not only limit the transmit rate of an application in times of congestion, but to also provide the application with the most throughput it's entitled to. Congestion control is difficult to get right, and many application-level implementations greatly simplify things, often leading to various inefficiencies. As mentioned above, many applications simply set a peak data rate limit. This makes extra capacity unavailable, and causes unnecessary congestion in the network when sufficient capacity isn't available.

By using DCCP, a multi-player game application could completely offload congestion control considerations, and benefit from a much more complete implementation than would normally be provided at the application layer, including support for ECN and the ability to use all of the available bandwidth. The application's flow control algorithm could be to simply write a message whenever the DCCP socket is ready. DCCP would manage probing the network for available capacity and backing off in the presence of network congestion or server load (via the slow receiver indications).

The CCID to use would depend on tradeoffs between smoothness and throughput. CCID3 would provide less dramatic changes in the rate of state transmissions (and receptions), perhaps eliminating abrupt changes in the display update rate. CCID2 would more rapidly consume available capacity, perhaps leading to noticeable changes in the update rate, but more updates at the peak.

3.1.1 Idleness Problems

Often the amount of data that needs to be transmitted depends on the amount of user activity at a given point. When the user is idle, little needs to be sent. When the user is active, a great deal can be sent. Often the transition between these two states is immediate.

Even though it has long been known to be harmful for the Internet [CONGAVOID], one of the simplifications often made by applications implementing congestion control is to ignore the issues of fast startup or restart after idle. Often these applications simply start transmitting immediately at a high rate, without ramping up.

The DCCP congestion control algorithms enforce slow-start and restart. For applications used to ignoring these issues, however, this could lead to unexpected behavior. In particular, the application could appear less responsive to quick shifts in the activity rate.

3.2 Partial Reliability

An application that implements reliability on top of UDP for some (or all) of its packets must implement application level acknowledgements and retransmit timers. The most efficient implementations of retransmission timers take the current RTT into account. This is often difficult to do at the application layer, so RTT is often ignored in favor of a pre-configured value likely to be "large enough". This can lead to long delays in the face of lost packets.

Although DCCP doesn't implement retransmissions, the nature of congestion control forces it to implement most of what is required for reliable delivery. An application using DCCP could simplify (and improve) its implementation of reliability by offloading some of the functions to DCCP.

By its nature, congestion control must know if packets are lost. For CCID2, which requires the use of the Ack Vector option, it even knows specifically which packets are lost. If a DCCP implementation were to make this information available to an application, it could ease the burden of implementing reliability.

Say, for example, that a DCCP allowed the application to request delivery indication for certain packets. DCCP would inform the application when either the packet was acknowledged, or DCCP had decided that the packet was dropped. The application would need to save the packet for retransmission, but it wouldn't need to maintain a retransmission timer or implement application-layer acknowledgements.

Because DCCP maintains connection state, such as current RTT, it can often make this decision more efficiently than the application. By tying the retransmission timeout to the current RTT, lost packets can be retransmitted sooner, with less chance of unneeded retransmissions. In addition, application layer acknowledgements would often be redundant with DCCP acknowledgements.

When CCID3 is used, the application could request the use of Ack Vector options and receive the same service. On the other hand, the normal use in CCID3 of the Loss Event Rate option might provide sufficient information. In this case the reliable packets would be considered delivered if there were no losses in the interval that contained the packet, and not delivered if there were losses. This might lead to some unnecessary retransmissions, but the acknowledgement savings (Loss Event Rate options are smaller and simpler to deal with than Ack Vector options) might make up for it.

3.3 Sequence Numbers

Game applications using UDP normally include sequence numbers in their application headers to allow the receiver to detect packet reordering. These application layer sequence numbers would often be redundant with the DCCP sequence numbers. If a DCCP application made the DCCP sequence numbers available to receiving applications, the application could determine the transmission order. Note that, since DCCP sequence numbers increase for Ack packets as well as Data packets, the application wouldn't be able to infer missing packets from holes in the delivered sequence.

4. Miscellaneous Capabilities

This section covers DCCP capabilities not covered earlier that might be unfamiliar to a developer accustomed to UDP communications.

4.1 Path MTU Discovery

DCCP mandates the use of Path Maximum Transfer Unit (PMTU) discovery. In general, an application will not be allowed to transmit a packet larger than the currently known PMTU (or the maximum packet size allowed by the CCID in effect). Applications can be allowed to override this restriction (at least for PMTU) and send packets larger than the PMTU (but not larger than the CCID maximum). These large packets will of course be fragmented as they travel through the network. However, since most applications would like to avoid fragmenting packets, the default DCCP behavior fits nicely.

UDP applications that wish to avoid packet fragmentation usually limit the size of their packets to 576 bytes, even though most connection paths can support much larger sizes. With PMTU discovery in DCCP, larger packet sizes can be used safely.

PMTU discovery in DCCP is based on the use of ICMP "packet too big" messages, as defined in [RFC 1911]. The PMTU is initially set to the MTU of the first-hop interface. Transmitted packets have the "don't fragment" bit set in the IP header. If an interface with a lower MTU is encountered, the router sends an ICMP Destination Unreachable message, with cause set to "fragmentation needed and DF set", to the originator (colloquially referred to as a "packet too big" message). When the originating DCCP receives the packet too big message, it adjusts the PMTU according to [RFC 1911].

Of course the packet that caused the packet too big message will be dropped, and since DCCP provides an unreliable service, it won't be retransmitted. There are also several other known issues with [RFC 1911]-style PMTU discovery (e.g., some firewalls block incoming ICMP messages). Applications that would like to use the biggest packets possible might want to consider an application-level supplement to DCCP's PMTU discovery (e.g, send an initial packet stream in various sizes and choose packet size based on the largest packet acknowledged by the receiver).

DCCP implementations should consider providing this procedure for applications, based on sending gratuitous DCCP-Sync packets padded out to various sizes with Padding Options. The receiving DCCP will acknowledge the DCCP-Sync packets that got through (with normal-sized DCCP-Sync packets of its own), and the PMTU can be set to the largest acknowledged packet.

4.2 Mobility and Multihoming

DCCP supports the ability to change the location of a connection endpoint (IP address and port) during the connection. This capability provides simple support for mobile hosts or high-availability applications. The use of mobility must be negotiated at connection setup, but the set of possible new addresses is not constrained at that time.

When an endpoint would like to move, it sends a DCCP-Move packet. The source address in the IP header is the new address, and the source port in the DCCP header is the new port. The packet also includes values for Mobility ID and Identification Option that were negotiated at connection setup. If the other end accepts the change it sends a DCCP-Ack to the new address/port; otherwise it sends a DCCP-Ack to the old address/port. The DCCP-Move packet also may include user data, but usually data transfer is interrupted until the move is complete. An endpoint may move any number of times during a connection. However, after each move, a new Mobility ID must be negotiated.

4.3 Partial Checksums and Payload Checksums

Applications that would rather receive corrupted data than have it discarded by the network can use the DCCP partial checksum capability. This capability allows the sender to specify how much of the user data should be covered by the DCCP header checksum. Options include all of the user data, none, or up to 14 32-words (this final option is considered experimental). DCCP implementations should allow senders to specify the checksum coverage per packet and allow receivers to specify the minimum checksum coverage they'd like to receive. Defaults for both of these values should be to cover all data.

Of course this is really only useful if there are link layers that detect ~~a~~ that a DCCP packet is being sent and provide strong error checking only for the portion of the packet covered by the DCCP checksum. Only time will tell if any link layers will implement this.

Applications that would like greater protection on their data than the Internet checksum provides can use the Payload Checksum option. This option contains a CRC-32 checksum of the payload data only.

4.4 ECN Support

DCCP supports the use of Explicit Congestion Notification [RFC 3168]. ECN allows routers to mark packets that have experienced congestion, rather than drop them. This provides DCCP with the capability to adjust to network resources before packets are lost. This capability is usually impossible to implement at the application layer, due to kernel restraints often imposed on setting and receiving ECN-marks in packets.

4.5 Slow Receiver Option

The Slow Receiver Option is sent by a DCCP when the receiving application indicates that it is unable to keep up with the incoming data rate. A DCCP that receives a Slow Receiver Option is required to not increase its sending rate for one RTT, and should indicate to its application that the option has been received.

This perhaps can be used by applications as a lightweight application-level flow control. If application data needs to be paced outside of congestion control, an overflow at the receiver can be communicated back with the Slow Receiver Option, instead of using an entire application-level message. Upon receiving a Slow Receiver Option, the transmitting application can make whatever adjustment is necessary. The application will of course be constrained by the requirement on DCCP to not allow the transmit rate to increase for one RTT.

4.6 Detecting Lost Application Data

In general, DCCP knows when packets are lost, but since data and acknowledgment packets use the same sequence number space, it can't tell whether or not application data has been lost. Applications that want to detect data loss at the receiver could implement their own application-layer sequence numbers for this purpose, but the DCCP NDP Count Feature and Option could allow the application to push this to DCCP.

When the NDP (stands for Non-Data Packets) Count Feature is in use, DCCP sends an NDP Count Option on every packet whose immediate predecessor was a non-data packet (mostly DCCP-Acks). With the NDP Count Options, the receiving DCCP can determine whether or not a hole in the received sequence numbers represents lost data. The receiving DCCP could then inform the application that data was lost.

Note that another use for application-layer sequence numbers is to reorder packets received out of transmission order. However, as mentioned in section 3.3, the DCCP sequence numbers are adequate for this.

5. Security Considerations

DCCP includes several non-cryptographic security features designed to limit vulnerability to some common Denial of Service (DoS) and other attacks. In UDP-based applications these capabilities would need to be implemented at the application layer, but are often ignored.

To insert packets into a DCCP connection, an attacker must guess proper sequence numbers. With randomly chosen initial sequence numbers, an attacker must snoop the connection to have any reasonable chance of success. Other mechanisms (such as ignoring invalid DCCP-Move packets) prevent leakage of information to attackers.

To hijack a connection (with DCCP-Move packets), an attacker must know the Mobility ID and Identification Option in use. Again, without snooping the connection, there is little chance of guessing these accurately.

The Init Cookie Option allows a server to delay holding state for a connection until the client has proved its aliveness. Basically, the server responds to a DCCP-Request packet with a DCCP-Response packet that contains an Init Cookie Option. This Init Cookie Option wraps up all information necessary for the connection to proceed in an encrypted and authenticated package. After sending the DCCP-

Response, the server needn't remember that a connection handshake is in progress. The client responds to the DCCP-Response with a DCCP-Ack that includes the Init Cookie. The server can then instantiate the necessary connection state.

6. IANA Considerations

There are no IANA actions required for this document.

7. Thanks

Thanks to Damon Lanphear for the original, API-centric, version of the user guide, the AVT working group, especially Philippe Gentric and Brian Rosen, for comments on the streaming-media-centric earlier version of this, ~~and~~ Bart Whitebrook and Vladimir Moltchanov for information on game programming, and Jukka Manner for comments.

8. Informative References

[DCCP] E. Kohler, M. Handley, S. Floyd, J. Padhye, Datagram Congestion Control Protocol (DCCP), February 2004, draft-ietf-dccp-spec-06.txt, work in progress.

[CCID2] S. Floyd, E. Kohler, Profile for DCCP Congestion Control 2: TCP-Like Congestion Control, February 2004, draft-ietf-dccp-ccid2-05.txt, work in progress.

[CCID3] S. Floyd, E. Kohler, J. Padhye, Profile for DCCP Congestion Control 3: TFRC Congestion Control, February 2004, draft-ietf-dccp-ccid3-04.txt, work in progress.

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

[RFC 768] J. Postel, User Datagram Protocol, August 1980, RFC 768.

[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, January 2004, draft-gentric-mmusic-stream-switching-01.txt, work in progress.

~~[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

[MMPGRID] Butterfly.net: Powering Next-Generation Gaming with Computing On-Demand, http://www.butterfly.net/platform/technology/idc.pdf

[CONGAVOID] V. Jacobson, M. Karels, Congestion Avoidance and Control, November 1988, ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z

[RFC 1911] J. Mogul, S. Deering, Path MTU Discovery, February 1996, RFC 1911

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

[RFC 3517] E. Blanton, M. Allman, K. Fall, L. Wang, A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP, April 2003, RFC 3517

[RFC 3550] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, July 2003, RFC 3550

[XTIME] ITU-T: Series G: Transmission Systems and Media, Digital Systems and Networks, Recommendation G.114, One-way Transmission Time, May 2000

9. Author's Address

Tom Phelan
Sonus Networks
5 Carlisle Rd.
Westford, MA USA 01886
Phone: 978-681-8456
Email: tphelan@sonusnet.com

10. Full Copyright Statement