Internet Engineering Task Force
INTERNET-DRAFT
draft-floyd-dcp-problem-01.ps

Sally Floyd
Mark Handley
Eddie Kohler
ICIR

26 August 2002
Expires: February 2003

Problem Statement for DCCP

Status of this Document

Abstract

This document gives the problem statement underlying the development of an unreliable transport protocol incorporating end-to-end congestion control. This is also the problem statement underlying the development of DCCP, the Datagram Congestion Control Protocol. DCCP implements a congestion-controlled, unreliable flow of datagrams suitable for use by applications such as streaming media or on-line games.

Table of Contents

1. Introduction 3
2. Problem Space 3
  2.1. Congestion Control for Unreliable Transfer 4
  2.2. Overhead 5
  2.3. Firewall Traversal 5
  2.4. Parameter Negotiation. 6
3. Solution Space for Congestion Control of Unreliable Flows 6
  3.1. Providing Congestion Control Above UDP. 6
    3.1.1. The Burden on the Application Designer 6
    3.1.2. Difficulties with ECN 7
    3.1.3. The Evasion of Congestion Control 7
  3.2. Providing Congestion Control Below UDP 8
    3.2.1. Case 1: Congestion Feedback at the Application 8
    3.2.2. Case 2: Congestion Feedback at a Layer below UDP. 8
  3.3. Providing Congestion Control at the Transport Layer 9
    3.3.1. Modifying TCP? 9
    3.3.2. Unreliable Variants of SCTP? 9
    3.3.3. Modifying RTP? 10
    3.3.4. Designing a New Transport Protocol 10
4. Selling Congestion Control to Reluctant Applications 11
5. Additional Design Considerations 11
6. Summary of Recommendations 12
7. References 12
8. Authors' Addresses. 13

1. Introduction

Historically, the great majority of Internet unicast traffic has used congestion-controlled TCP, with UDP making up most of the remainder. UDP has mainly been used for short, request-response transfers, like DNS and SNMP, that wish to avoid TCP's three-way handshake, retransmission, and/or stateful connections. UDP also avoids TCP's built-in end-to-end congestion control, and UDP applications tended not to implement their own congestion control. However, since UDP traffic volume was small relative to congestion-controlled TCP flows, the network didn't collapse.

Recent years have seen the growth of applications that use UDP in a different way. These applications, including RealAudio, Internet telephony, and on-line games such as Half Life, Quake, and Starcraft, share a preference for timeliness over reliability. TCP can introduce arbitrary delay because of its reliability and in-order delivery requirements; thus, the applications use UDP instead. This growth of long-lived non-congestion-controlled traffic, relative to congestion-controlled traffic, poses a real threat to the overall health of the Internet.

Applications could implement their own congestion control mechanisms on a case-by-case basis, with encouragement from the IETF. Some already do this. However, experience shows that congestion control is difficult to get right, and many application writers would like to avoid reinventing this particular wheel. We believe that a new protocol is needed, one that combines unreliable datagram delivery with built-in congestion control. This protocol would act as an enabling technology: existing and new applications could easily use it to transfer timely data without destabilizing the Internet.

This document provides a problem statement for such a protocol. We list the properties the protocol should have, then explain why those properties are necessary. We also describe why a new protocol is the best solution for the more general problem of bringing congestion control to unreliable flows of unicast datagrams.

This problem statement began as a formalization of the goals of DCCP, a proposed protocol already described in several Internet-Drafts. We intended DCCP to satisfy the problem statement laid out here. However, we believe the problem should be solved whether or not the chosen solution is DCCP, and the DCCP drafts should be judged based on this document or its successors. Nevertheless, for convenience, we write "DCCP" to mean "any protocol that satisfies this problem statement".

2. Problem Space

We perceive a number of problems related to the use of unreliable data flows in the Internet. The major issues are:

- The potential for non-congestion-controlled datagram flows to cause congestion collapse of the network.
- The difficulty of correctly implementing effective congestion control mechanisms for unreliable datagram flows.
- The lack of a standard solution for performing reliable congestion feedback for unreliable data flows.
- The need for a standard solution for negotiating ECN usage for unreliable flows.
- The need for applications to have a choice of TCP-friendly congestion control mechanisms.

While undoubtedly some application writers would prefer not to perform congestion control at all for unreliable flows, we expect that most responsible application writers would use congestion control if it were available in a standard, easy to use form.

More minor problems are:

- The difficulty of deploying applications using UDP-based flows in the presence of firewalls.
- The desire to have a single way to negotiate congestion control parameters for unreliable flows, independently of the signalling protocol used to set up the flow.
- The desire for low per-packet byte overhead.

The subsections below discuss these problems of providing congestion control, traversing firewalls, and negotiating parameters in more detail. A separate subsection also discusses the problem of minimizing the overhead of packet headers.

2.1. Congestion Control for Unreliable Transfer

This project aims to bring easy-to-use congestion control mechanisms to applications that generate long-lived, large flows of unreliable datagrams, such as RealAudio, Internet telephony, and multiplayer games. Our motivation is to avoid congestion collapse. (Request-response applications, such as DNS, SNMP, and SIP, are not our concern; their short flows don't cause congestion in practice, and any congestion control mechanism would take effect between flows, not within a single end-to-end transfer of information.) However, before designing a congestion control mechanism for these applications, we must understand why they use unreliable datagrams in the first place, lest we destroy the very properties they require.

There are several reasons why protocols currently use UDP instead of TCP, amongst them:

- Startup Delay: they wish to avoid the delay of a three-way handshake.
- Statelessness: they wish to avoid holding connection state, and the potential state-holding attacks that come with this.
- Trading of Reliability against Timing: the data being sent is timely in the sense that if it is not delivered by some deadline (typically a small number of RTTs) then the data will not be useful at the receiver.

Of these, our target applications -- media transfer, games, and the like -- mostly care about having control over the tradeoff between timing and reliability. Such applications use UDP because when they send a datagram, they wish to send the most appropriate data in that datagram. If the datagram is lost, they may or may not resend the same data, depending on whether the data will still be useful at the receiver. Data may no longer be useful for many reasons:

- In a telephony or streaming video session, data in a packet comprises a timeslice of a continuous stream. Once a timeslice has been played out, the next timeslice is required immediately. If the data comprising that timeslice arrives at some later time, then it is no longer useful. Such applications can cope with masking the effects of missing packets to some extent, so when the sender transmits its next packet, it is important for it to only send data that has a good chance of arriving in time for its playout.
- In a interactive game or VR session, position information is transient. If a datagram containing position information is lost, resending the old position does not usually make sense -- rather, every position information datagram should contain the latest position information.

In a congestion-controlled flow, the allowed packet sending rate depends on measured network congestion. Thus, applications must give up some control to the congestion control mechanism, which determines precisely when packets can be sent. However, applications could still decide, *at transmission time,* which information to put in a packet. TCP doesn't allow control over this; these applications demand it.

Often, these applications (especially games and telephony applications) work on very short playout timescales. Whilst they are usually able to adjust their transmission rate based on congestion feedback, they do have constraints on how this adaptation can be performed so that it has minimal impact on the quality of the session. Thus, they tend to need some control over the short-term dynamics of the congestion control algorithm, whilst being fair to other traffic on medium timescales. This control includes, but is not limited to, some influence on which congestion control algorithm should be used -- for example, TFRC rather than strict TCP-like congestion control.

2.2. Overhead

The applications we are concerned with often send compressed data, or send frequent small packets. For example, when internet telephony or streaming media are used over low-bandwidth modem links, highly compressing the payload data is essential. For internet telephony applications and for games, the requirement is for low delay, and hence small packets are sent frequently.

For example, a telephony application sending a 20Kb/s packet stream wanting moderately low delay may send a packet every 20ms, and so each packet would be only 50 bytes. Of this, 20 bytes is taken up by the IP header, leaving only 30 bytes for the transport header and payload. Of course this is a relatively extreme example, but it serves to illustrate the degree to which some of these applications care that the transport protocol is low overhead.

In some cases the correct solution would be to use link-based packet header compression to compress the packet headers, although we cannot guarantee the availability of such compression schemes on any particular link.

The delay of data until after the completion of a handshake also represents potentially unnecessary overhead. A new protocol might therefore allow senders to include some data on their initial datagrams.

2.3. Firewall Traversal

Applications requiring a flow of unreliable datagrams currently tend to use signalling protocols such as RTSP, SIP and H.323 in conjunction with UDP for the data flow. The initial setup request uses a signalling protocol to locate the correct remote end-system for the data flow, sometimes being redirected or relayed to other machines, before the data flow is established.

As UDP flows contain no explicit setup and teardown, it is hard for firewalls to handle them correctly. Typically the firewall needs to parse RTSP, SIP and H.323 to obtain the information necessary to open a hole in the firewall. Alternatively, for bi-directional flows, the firewall can open a bi-directional hole if it receives a UDP packet from inside the firewall, but in this case the firewall can't easily know when to close the hole again.

While we do not consider these to be major problems, they are nonetheless issues that application designers face. Currently streaming media players such as RealPlayer attempt UDP first, and then switch to TCP if UDP is not successful. Streaming media over TCP is undesirable, and can result in the receiver needing to temporarily halt playout while it "rebuffers" data. Telephony applications don't even have this option.

2.4. Parameter Negotiation

Different applications have different requirements of congestion control, which may map into different congestion feedback. Examples are the ECN capability, and desired congestion control dynamics, which in turn determines the choice of congestion control algorithm and the form of feedback information required. Such parameters need to be reliably negotiated before congestion control can function correctly.

While this negotiation could be performed using signalling protocols such as SIP, RTSP and H.323, it would be desirable to have a single standard way of negotiating these transport parameters. This is of particular importance with ECN, where sending ECN-marked packets to a non-ECN-capable receiver can cause significant congestion problems to other flows. We discuss the ECN issue in more detail below.

3. Solution Space for Congestion Control of Unreliable Flows

We thus want to provide congestion control for unreliable flows, providing both ECN and the choice of different forms of congestion control, and providing moderate overhead in terms of packet size, state, and CPU processing. There are a number of options for providing end-to-end congestion control for the unicast traffic that currently uses UDP, in terms of the layer that provides the congestion control mechanism:

- Congestion control above UDP.
- Congestion control below UDP.
- Congestion control at the transport layer as an alternative to UDP.

We explore these alternatives in the sections below. The concerns from the discussions below have convinced us that the best way to provide congestion control for unreliable flows is to provide congestion control at the transport layer, as an alternative to the use of UDP and TCP.

3.1. Providing Congestion Control Above UDP

One possibility would be to provide congestion control at the application layer, or at some other layer above UDP. Implementing congestion control at a layer above UDP would allow the congestion control mechanism to be closely integrated with the application itself.

3.1.1. The Burden on the Application Designer

A key disadvantage of providing congestion control above UDP is that it places an unnecessary burden on the application-level designer, who might be just as happy to use the congestion control provided by a lower layer. If the application can rely on a lower layer that gives a choice between TCP-like or TFRC-like congestion control, and that offers ECN, then this might be highly satisfactory to many application designers.

The long history of debugging TCP implementations [RFC 2525] [TBIT] makes the difficulties in implementing end-to-end congestion control abundantly clear. It is clearly more robust for congestion control to be provided for the application by a lower layer. In rare cases there might be compelling reasons for the congestion control mechanism to be implemented in the application itself, but we do not expect this to be the general case. For example, applications that use RTP over UDP might be just as happy if RTP itself implemented end-to-end congestion control. (See Section 3.3.3 for more discussion of RTP.)

In addition to congestion control issues, we also note the problems with firewall traversal and parameter negotiation discussed in sections 2.3 and 2.4. Implementing on top of UDP requires that the application designer also address these issues.

3.1.2. Difficulties with ECN

A second problem of providing congestion control above UDP is that it would require either giving up the use of ECN, or giving the application direct control over setting and reading the ECN field in the IP header. Giving up the use of ECN would be problematic, since ECN can be particularly useful for unreliable flows, where a dropped packet will not be retransmitted by the data sender.

With the development of the ECN nonce, ECN can also be useful even in the absence of ECN support from the network. The data sender can use the ECN nonce, along with feedback from the data receiver, to verify that the data receiver is correctly reporting all lost packets. This use of ECN can be particularly useful for an application using unreliable delivery, where the receiver might otherwise have little incentive to report lost packets.

In order to allow the use of ECN by a layer above UDP, the UDP socket would have to allow the application to control the ECN field in the IP header. In particular, the UDP socket would have to allow the application to specify whether or not the ECN-Capable Transport (ECT) codepoints should be set in the ECN field of the IP header.

The ECT codepoint can only safely be set in the packet header of a UDP packet if the following is guaranteed:

- If the Congestion Experienced (CE) codepoint is set by a router, the receiving UDP will pass that CE status to the receiving application at the data receiver, and:
- Upon receiving a packet that had the CE codepoint set, the receiving application will take the appropriate congestion control action, such as informing the data sender.

However, the UDP implementation at the data sender has no way of knowing if the UDP implementation at the data receiver has been upgraded to pass a CE status up to the receiving application, let alone whether or not the application will use the conformant end-to-end congestion control that goes along with use of ECN.

In the absence of the widespread deployment of mechanisms in routers to detect flows that are not using conformant congestion control, allowing applications arbitrary control of the ECT codepoints for UDP packets would seem like an unnecessary opportunity for applications to use ECN while evading the use of end-to-end congestion control. Thus, there is an inherent "chicken-and-egg" problem of whether first to deploy policing mechanisms in routers, or first to enable the use of ECN by UDP flows. Without the policing mechanisms in routers, we would not advise adding ECN-capability to UDP sockets at this time.

3.1.3. The Evasion of Congestion Control

A third problem of providing congestion control above UDP is that relying on congestion control at the application level makes it somewhat easier for some users to evade end-to-end congestion control. We do not claim that a transport protocol such as DCCP would always be implemented in the kernel, and do not attempt to evaluate the relative difficulty of modifying code inside the kernel vs. outside the kernel in any case. However, we believe that putting the congestion control at the transport level rather than at the application level makes it just slightly less likely that users will go to the trouble of modifying the code in order to avoid using end-to-end congestion control.

3.2. Providing Congestion Control Below UDP

Instead of providing congestion control above UDP, a second possibility would be to provide congestion control for unreliable applications at a layer below UDP, and the applications would use UDP as their transport protocol. Given that UDP does not itself provide sequence numbers or congestion feedback, there are two possible forms for this congestion feedback:

- (1) Feedback at the application: The application above UDP could provide sequence numbers and feedback to the sender, which would then communicate loss information to the congestion control mechanism. This is the approach currently standardized by the Congestion Manager [RFC 3124].
- (2) Feedback at the layer below UDP: The application could use UDP, and a protocol could be implemented using a shim header between IP and UDP to provide sequence number information for data packets and return feedback to the data sender. The original proposal for the Congestion Manager [Bala99] suggested providing this layer for applications that did not have their own feedback about dropped packets.

We discuss these two cases separately below.

3.2.1. Case 1: Congestion Feedback at the Application

In this case, the application provides sequence numbers and congestion feedback above UDP, but communicates that feedback to a congestion manager below UDP, which regulates when packets can be sent. This approach suffers from most of the problems described in section 3.1, namely forcing the application designer to reinvent the wheel each time for packet formats and parameter negotiation, and problems with ECN usage, firewalls and evasion.

It would avoid the application writer needing to implement the control part of the congestion control mechanism, but it is unclear how easily multiple congestion control algorithms (such as receiver-based TFRC) can be supported, given that the form of congestion feedback usually needs to be closely coupled to the congestion control algorithm being used.

3.2.2. Case 2: Congestion Feedback at a Layer below UDP

Providing feedback at a layer below UDP would require an additional packet header below UDP to carry sequence numbers in addition to the eight-byte header for UDP itself. Unless this header were an IP option (which is likely to cause problems for many IPv4 routers) then its presence would need to be indicated using a different IP protocol value from UDP. Thus, the packets would no longer look like UDP on the wire.

To use this mechanism most effectively, the semantics of the UDP socket API (Application Programming Interface) would also need changing, both to support a late decision on what to send, and to provide access to the sequence numbers to avoid the application needing to duplicate them for its own purposes. Thus, the socket API would no longer look like UDP in the end hosts. This would effectively be a new transport protocol.

Given these complications, it seems cleaner to actually design a new transport protocol, which also allows us to address the issues of firewall traversal, flow setup, and parameter negotiation. We note that any new transport protocol could also use a Congestion Manager approach to share congestion state between flows using the same congestion control algorithm, if this were deemed to be desirable.

3.3. Providing Congestion Control at the Transport Layer

The concerns from the discussions above have convinced us that the best way to provide congestion control to applications that currently use UDP is to provide congestion control at the transport layer, in a transport protocol used as an alternative to UDP. One advantage of providing end-to-end congestion control in an unreliable transport protocol is that it could be used easily by a wide range of the applications that currently use UDP, with minimal changes to the application itself. The application itself would not have to provide the congestion control mechanism, or even the feedback from the data receiver to the data sender about lost or marked packets.

The question then arises of whether to adapt TCP for use by unreliable applications, to use an unreliable variant of SCTP or a version of RTP with built-in congestion control, or to design a new transport protocol.

As we argue below, the desire for minimal overhead results in the design decision to use a transport protocol containing only the minimal necessary functionality, and to leave other functionality such as reliability, semi-reliability, or Forward Error Correction (FEC) to be layered on top.

3.3.1. Modifying TCP?

One alternative to be considered would be to create an unreliable variant of TCP, with the reliability layered on top for applications desiring reliable delivery. However, our requirement is not simply for TCP minus the in-order reliable delivery, but also for the application to be able to choose congestion control algorithms. TCP's feedback mechanism works well for TCP-like congestion control, but is inappropriate (or at the very least, inefficient) for TFRC. In addition, TCP sequence numbers are in bytes, which is not optimal for an unreliable datagram protocol. Finally, there is the issue of whether a modified TCP would require a new IP protocol number as well; a significantly modified TCP using the same IP protocol number could have unwanted interactions with some of the middleboxes already deployed in the network.

In conclusion, it seems best simply to leave TCP as it is, and to create a new congestion control protocol for unreliable transfer.

3.3.2. Unreliable Variants of SCTP?

SCTP was in part designed to accommodate multiple streams within a single end-to-end connection, modifying TCP's semantics of reliable, in-order delivery to allow out-of-order delivery. However, explicit support for multiple streams over a single flow at the transport layer is not necessary for an unreliable transport protocol such as DCCP, which of necessity allows out-of-order delivery. Because an unreliable transport does not need streams support, applications should not have to pay the penalties in terms of increased header size that accompany the use of streams in SCTP.

The basic underlying structure of the SCTP packet, into a common SCTP header followed by chunks for data, SACK information, and so on, is motivated by SCTP's goal of accommodating multiple streams, However, this use of chunks comes at the cost of an increased header size for packets, as each chunk must be aligned on 32-bit boundaries, and therefore requires a fixed-size 4-byte chunk header. For example, for a connection using ECN, SCTP includes separate control chunks for the Explicit Congestion Notification Echo and Congestion Window Reduced functions, with the ECNE and CWR chunks each requiring 8 bytes. As another example, the common header includes a 4-byte verification tag to validate the sender. The issue of a minimal header size is just one of the ways that the underlying framework of SCTP does not fit the needs of an unreliable transport as outlined in this document.

As a second concern, SCTP as currently specified uses TCP-like congestion control, and does not provide support for alternative congestion control algorithms such as TFRC that would be more attractive to some of the applications currently using UDP flows. Thus, the current version of SCTP would not meet the requirements for a choice between forms of end-to-end congestion control.

One could suggest adding support for alternative congestion control mechanisms as an option to SCTP, and adding a fully-unreliable variant that does not include the mechanisms for multiple streams. However, that path leads one to a choice between a ‘‘kitchen sink’’ protocol that tries to include options to accommodate all applications (i.e., the modified version of SCTP), and a clean and minimal protocol that provides only end-to-end congestion control and any other mechanisms that cannot be provided in a higher layer. We would argue against the ‘‘kitchen sink’’ approach in this case.

Applications that desire limited retransmission with multi-stream support, or that desire multi-homing support, might be good candidates for a semi-reliable version of SCTP such as U-SCTP. However, we believe that variants of SCTP are not suitable for fully-unreliable applications, or for the bulk of applications that today use UDP. We would argue that instead, an additional transport protocol is needed for unreliable transfer.

3.3.3. Modifying RTP?

Several of our target applications currently use RTP layered above UDP to transfer their data. Why not modify RTP to provide end-to-end congestion control?

As RTP lives above UDP, modifying it to support congestion control might encounter some of the problems described in Section 3.1. In particular, user-level RTP implementations would want access to ECN bits in UDP datagrams. It might be difficult or undesirable to allow that access for RTP, but not for other user-level programs.

Kernel implementations of RTP would not suffer from this problem. In the end, the argument against modifying RTP is the same as that against modifying SCTP: Some applications, such as the export of flow information from routers, need congestion control but don’t need much of RTP’s functionality. From these applications’ point of view, RTP would induce unnecessary overhead. Again, we would argue for a clean and minimal protocol focused on end-to-end congestion control.

RTP would commonly be used as a layer above any new transport protocol, however. The design of that new transport protocol should take this into account, either by avoiding undue duplication of information available in the RTP header, or by suggesting modifications to RTP.

3.3.4. Designing a New Transport Protocol

In the first half of this document we have argued for providing congestion control at the transport layer as an alternative to UDP, instead of relying on congestion control supplied only above or below UDP. In this section, we have examined the possibilities of modifying SCTP, modifying TCP, and designing a new transport protocol. In large part because of the requirement for unreliable transport, and for accommodating TFRC and well as TCP-like congestion control, we have concluded that modifications of SCTP or TCP are not the best answer, and that a new transport protocol is needed. Thus, we have argued for the need for a new transport protocol that offers unreliable delivery; accommodates TFRC as well as TCP-like congestion control; accommodates the use of ECN; and requires minimal overhead in packet size and in the state and CPU processing required at the data receiver.

This is the line of reasoning that has lead us to the development of DCCP. Both the problem statement presented in this document, and the detailed design decisions made in the design of DCCP, have been brought to the IETF for further feedback. This document addresses only the problem statement; the design decisions resulting from this problem statement will be addressed in a later document.

4. Selling Congestion Control to Reluctant Applications

The goal of this work is to provide general congestion control mechanisms that will actually be used by many of the applications that currently use UDP. This may include applications that are perfectly happy without end-to-end congestion control. Several of our design requirements follow from a desire to design and deploy a congestion-controlled protocol that is actually attractive to these "reluctant" applications. These include the use of Explicit Congestion Notification (ECN) and the ECN Nonce, which both provide positive benefit to the application itself; the choice between different forms of congestion control; and moderate overhead in the size of the packet header.

There will always be a few flows that are resistant to the use of end-to-end congestion control, preferring an environment where end-to-end congestion control is used by everyone else, but not by themselves. There has been substantial agreement [RFC 2309] [FF99] that in order to maintain the continued use of end-to-end congestion control, router mechanisms are needed to detect and penalize uncontrolled high-bandwidth flows in times of high congestion; these router mechanisms are colloquially known as ‘penalty boxes’. However, before undertaking a concerted effort towards the deployment of penalty boxes in the Internet, it seems reasonable, and more effective, to first make a concerted effort to make end-to-end congestion control easily available to applications currently using UDP.

5. Additional Design Considerations

This section mentions some additional design considerations that should be considered in designing a new transport protocol.

- Mobility: Mechanisms for multi-homing and mobility are one area of additional functionality that cannot necessarily be layered cleanly and effectively on top of a transport protocol. Thus, one outstanding design decision with any new transport protocol concerns whether to incorporate mechanisms for multi-homing and mobility into the protocol itself.
- Defense against DoS attacks and spoofing: A reliable handshake for connection setup and teardown offers protection against DoS and spoofing attacks. Mechanisms allowing a server to avoid holding any state for unacknowledged connection attempts or already-finished connections offers additional protection against DoS attacks. Thus, in designed a new transport protocol, even one designed to provide minimal functionality, the requirements for providing defense against DoS attacks and spoofing need to be considered.
- Interoperation with RTP: As Section 3.3.3 describes, attention should be paid to the changes that would be required by RTP in order to use any new protocol.
- API: Some functionality required by the protocol, or useful for applications using the protocol, may require the definition of new API mechanisms. Examples include allowing applications to decide what information to put in a packet at transmission timer, and providing applications with some information about packet sequence numbers.

- Consider general experiences with unicast transport: A Requirements for Unicast Transport/Sessions (ruts) BOF was held at the IETF meeting in December, 1998, with the goal of understanding the requirements of applications whose needs were not met by TCP [RUTS]. Not all of those unmet needs are relevant to or appropriate for a unicast, congestion-controlled, unreliable flow of datagrams designed for long-lived transfers. Some are, however, and any new protocol should address those needs, and other requirements derived from the community's experience. We believe that the DCCP problem statement addresses the relevant requirements brought up at the ruts BOF.

6. Summary of Recommendations

Our problem statement has discussed the need for implementing congestion control for unreliable flows. Additional problems concern the need for low overhead, the problems of firewall traversal, and the need for reliable parameter negotiation. Our consideration of the problem statement has resulted in the following general recommendations:

- A unicast transport protocol for unreliable datagrams should be developed, including mandatory, built-in congestion control.
- The protocol must provide a set of congestion control mechanisms from which the application may choose. These mechanisms should include, at minimum, TCP-like congestion control and a more slowly-responding congestion control such as TFRC.
- Important features of the connection, such as the congestion control mechanism in use, should be reliably negotiated by both endpoints.
- Support for ECN should be included.
- The overhead must be low, in terms of both packet size and protocol complexity.
- Some DoS protection for servers must be included. In particular, servers can make themselves invulnerable to spoofed connection attacks ("SYN floods").
- Connection setup and teardown must use explicit handshakes, facilitating transmission through stateful firewalls.

If there is a consensus about the need for a new unicast transport protocol for unreliable datagrams, then the next step can be the consideration of more detailed architectural issues.

7. References

[Bala99] H. Balakrishnan, H. Rahul, and S. Seshan, An Integrated Congestion Management Architecture for Internet Hosts, SIGCOMM, Sept. 1999.

[MEASWEB] Ramon Caceres and Sally Floyd, Measurement Studies of End-to-End Congestion Control in the Internet, Web Page, 2001.

[FF99] S. Floyd and K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet, IEEE/ACM Transactions on Networking, August 1999.

[MC01] S. McCreary and K.C. Claffy, Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange, ITC Specialist Seminar, 2001. URL "http://www.caida.org/outreach/papers/2000/AIX0005/".

[RFC 2026] S. Bradner. The Internet Standards Process -- Revision 3. RFC 2026.

[RFC 2481] K. Ramakrishnan, S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481.

[RFC 1191] J. C. Mogul, S. E. Deering. Path MTU Discovery. RFC 1191.

[RFC 2309] B. Braden et al., Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309, April 1998.

[RFC 2525] V. Paxson et al., Known TCP Implementation Problems, RFC 2525, March 1999.

[RFC 2914] S. Floyd. Congestion Control Principles. RFC 2914, Sept. 2000.

[RFC 2960] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson. Stream Control Transmission Protocol. RFC 2960.

[RFC 3124] H. Balakrishnan, S. Seshan. The Congestion Manager. RFC 3124.

[RUTS] Requirements for Unicast Transport/Sessions (ruts) BOF, Dec. 7, 1998. URL "http://www.ietf.org/proceedings/98dec/43rd-ietf-98dec-142.html".

[TBIT] J. Padhye and S. Floyd, Identifying the TCP Behavior of Web Servers, SIGCOMM 2001.

[WES01] David Wetherall, David Ely, Neil Spring. Robust ECN Signaling with Nonces. draft-ietf-tsvwg-tcp-nonce-03.txt, work in progress, April 2002.

8. Authors' Addresses

Sally Floyd <floyd@icir.org>
Mark Handley <mjh@icir.org>
Eddie Kohler <kohler@icir.org>

ICSI Center for Internet Research (ICIR),
International Computer Science Institute,
1947 Center Street, Suite 600
Berkeley, CA 94704.