

http://researchcommons.waikato.ac.nz/

Research Commons at the University of Waikato

Copyright Statement:

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

The thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of the thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from the thesis.

Improving the Quality of Real Time Media Applications through Sending the Best Packet Next

A thesis

submitted in fulfilment

of the requirements for the Degree

of

Doctor of Philosophy

at

The University of Waikato

by

IAN MCDONALD



THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2013

Abstract

Real time media applications such as video conferencing are increasing in usage. These bandwidth intensive applications put high demands on a network and often the quality experienced by the user is sub-optimal.

In a traditional network stack, data from an application is transmitted in the order that it is received. This thesis proposes a scheme called "Send the Best Packet Next (SBPN)" where the most important data is transmitted first and data that will not reach the receiver before an expiry time is not transmitted. In SBPN the packet priority and expiry time are added to a packet and used in conjunction with the Round Trip Time (RTT) to determine whether packets are sent, and in which order that they are sent. For example, it has been shown that audio is more important to users than video in video conferencing. SBPN could be considered to be Quality of Service (QoS) that is within an application data stream. This is in comparison to network routers that provide QoS to whole streams such as Voice over IP (VoIP), but do not differentiate between data items within the stream or which data gets transmitted by the end nodes. Implementation of SBPN can be done on the server only, so that much of the benefit for one way transmission (e.g. live television) can be gained without requiring existing clients to be changed.

SBPN was implemented in a Linux kernel on top of Datagram Congestion Control Protocol (DCCP) and compared to existing solutions. This showed real improvement in the measured quality of audio with a maximum improvement of 15% in selected test scenarios.

Acknowledgements

I would like to thank Richard Nelson for his support and guidance as my primary supervisor and for Matthew Luckie, Tony McGregor, Matt Jones and Alan Holt who all acted as supervisors during different periods of my study.

Without the help of Arnaldo Carvalho de Melo, Shane Alcock, Patrick McManus, Luleå University of Technology and others before them I could not have had a working code base for DCCP.

Gerrit Renker gave me advice on both code and on my research. Thank you Gerrit.

There were others also who reviewed papers and my thesis and gave many helpful suggestions along the way - Sam Jansen, Eddie Kohler, Gorry Fairhurst, Andreas Löf, Craig Box, Bill Rogers and Sally Jo Cunningham.

I would also like to thank all the members of the WAND research group at the University of Waikato who supported me in many different ways.

I would like to thank Jayne my wife, for her support in so many ways - without it I wouldn't have been able to undertake this work and she convinced me to keep going despite all the difficulties life throws out.

This research was funded by the WAND network research group and the Computer Science department at the University of Waikato.

Contents

1 Introduction **1**

1.1 The problem: Increasing media traffic and congestion control . 1

1.2 The proposed solution 3

1.3 Overview of thesis 4

1.4 Contribution of this work 6

2 Background **8**

2.1 Introduction 8

2.2 Congestion 9

2.2.1 Media applications and congestion 10

2.2.2 Approaches to Ensuring Service Quality 12

2.3 Protocols 13

2.3.1 TCP 13

2.3.2 UDP 18

2.3.3 RTP 19

2.3.4 SCTP 20

2.3.5 DCCP 21

2.4 Network protocol interaction with media applications 23

2.4.1 The use of audio and video layers with congestion . . . 25

2.4.2 Transmission buffers 28

2.5 Summary 29

3 Proposed solution **32**

3.1 Introduction 32

3.2 Use of DCCP 33

3.2.1 Rationale 33

3.2.2 TFRC congestion control 34

3.3 Sending the Best Packet Next 35

3.3.1 Not all packets are created equal 37

3.3.2 Passing information to the operating system 38

3.3.3 Allow altered packet sending and discard order 38

3.4 SBPN1 algorithm 39

3.5 SBPN2 . . . 40
3.6 Summary . . . 41

4 Implementation **43**
4.1 Introduction . . . 43
4.2 Implementation of DCCP . . . 44
4.2.1 History . . . 44
4.2.2 DCCP Tools . . . 46
4.2.3 Implementation challenges . . . 46
4.3 Implementation of SBPN in Linux DCCP . . . 48
4.4 Video conference model . . . 50
4.4.1 Data captured . . . 50
4.4.2 Ekiga results . . . 51
4.4.3 MSN Messenger results . . . 52
4.4.4 Skype results . . . 53
4.4.5 Traffic model . . . 54
4.5 Building of test framework . . . 55
4.5.1 Netem . . . 55
4.5.2 Test applications . . . 56
4.6 Measuring audio quality . . . 58
4.6.1 On time arrival . . . 59
4.6.2 Mean Opinion Score . . . 60
4.7 Summary . . . 62

5 Results from testing SBPN **64**
5.1 Introduction . . . 64
5.2 Testing with loss . . . 65
5.2.1 SBPN1 . . . 65
5.2.2 Testing SBPN2 . . . 71
5.2.3 Analysing packets transmitted . . . 72
5.3 Testing with congestion . . . 75
5.3.1 Testing SBPN2 with congestion . . . 75
5.3.2 Testing differing queue lengths with SBPN2 . . . 77
5.3.3 Testing SBPN2 with varying RTTs . . . 81
5.4 Summary . . . 85

6 Faster Restart and LIFO **87**
6.1 Introduction . . . 87
6.2 TFRC Faster Restart . . . 87
6.2.1 Implementation of TFRC Faster Restart . . . 88
6.2.2 Results from TFRC Faster Restart . . . 88

6.3 LIFO 90
6.4 Summary 94

7 Ring buffers **95**
7.1 Introduction 95
7.2 Ring buffers theory 96
7.3 Results of Ring Buffer compared to SBPN2 97
7.4 Combining SBPN2 and Ring buffers 101
7.5 Comparing Ring Buffers to SBPN2 with Ring Buffers 106
7.6 Summary 111

8 Conclusion **113**
8.1 Introduction 113
8.2 Practical work 113
8.3 Best results 114
8.4 Further work and future research 116
8.4.1 Changing bandwidth 116
8.4.2 Modifying applications 116
8.4.3 Sending all packets 117
8.4.4 Testing with multiple layers 118
8.4.5 Subjective measurement 118
8.4.6 Other areas of research 118
8.5 Summary of Contributions 119

Appendices **121**

A Problem statement for DCCP **121**

B Acronyms **143**

List of Figures

2.1 Diagram from Ars Technica [51] showing WHDI protecting data in higher priority layers 28

3.1 The expiry algorithm for SBPN1 39

3.2 The expiry algorithm for SBPN2 41

4.1 sendmsg system call function prototype in Linux 48

4.2 msghdr structure for sendmsg system call 49

4.3 dccp_prio structure passed in msg_control field 50

4.4 Test network for SBPN testing - Netem installed on middle box 55

4.5 Diagram illustrating on-time arrival 59

5.1 Testing with loss - 80 ms RTT, unbounded queue 65

5.2 Testing with loss - 80 ms RTT, unbounded queue 67

5.3 Testing with loss - 80 ms RTT, queue length 5 69

5.4 Testing with loss - 80 ms RTT, queue length 5 69

5.5 Timing of audio and video packet transmission 70

5.6 Testing with loss - 80 ms RTT, queue length 5 71

5.7 Testing with loss - 80 ms RTT, queue length 5 72

5.8 Testing with loss - 80 ms RTT, queue length 5 73

5.9 Testing with loss - 80 ms RTT, queue length 5 73

5.10 Difference in on time arrival - 80 ms RTT, queue length 5 . . . 74

5.11 Unmodified DCCP vs SBPN2 with congestion - 80 ms RTT, queue length 5 76

5.12 Unmodified DCCP vs SBPN2 with congestion - 80 ms RTT, queue length 5 76
5.13 Unmodified DCCP vs SBPN2 with congestion - 80 ms RTT, queue length 32 77
5.14 Unmodified DCCP vs SBPN2 with congestion - 80 ms RTT, queue length 32 78
5.15 Unmodified DCCP with congestion, 80 ms RTT, queue length 5 vs 32 79
5.16 Unmodified DCCP with congestion, 80 ms RTT, queue length 5 vs 32 79
5.17 SBPN2 with congestion, 80 ms RTT, queue length 5 vs 32 . . 80
5.18 SBPN2 with congestion, 80 ms RTT, queue length 5 vs 32 . . 80
5.19 Unmodified DCCP vs SBPN2 with congestion - 30 ms RTT, queue length 32 81
5.20 Unmodified DCCP vs SBPN2 with congestion - 30 ms RTT, queue length 32 82
5.21 Unmodified DCCP vs SBPN2 with congestion - 150 ms RTT, queue length 32 82
5.22 Unmodified DCCP vs SBPN2 with congestion - 150 ms RTT, queue length 32 83

6.1 Unmodified DCCP vs Faster Restart - 80 ms RTT, queue length 5 88
6.2 Unmodified DCCP vs Faster Restart - 80 ms RTT, queue length 5 89
6.3 SBPN2 vs LIFO priority queues - 80 ms RTT, queue length 5 91
6.4 SBPN2 vs LIFO priority queues - 80 ms RTT, queue length 5 91
6.5 SBPN2 vs LIFO queue - 80 ms RTT, queue length 5 92
6.6 SBPN2 vs LIFO queue - 80 ms RTT, queue length 5 92
6.7 LIFO priority queues vs LIFO queue - 80 ms RTT, queue length 5 93

7.1 Ring buffer pseudo code - insertion 97
7.2 SBPN2 vs Ring buffers - 80 ms RTT, queue length 5 98

7.3 SBPN2 vs Ring buffers - 80 ms RTT, queue length 5 98
7.4 SBPN2 vs Ring buffers - 80 ms RTT, queue length 32 100
7.5 SBPN2 vs Ring buffers - 80 ms RTT, queue length 32 100
7.6 Ring buffers - 80 ms RTT, queue length 5 vs 32 102
7.7 Ring buffers - 80 ms RTT, queue length 5 vs 32 102
7.8 SBPN2 vs SBPN2 with ring - 80 ms RTT, queue length 5 . . . 103
7.9 SBPN2 vs SBPN2 with ring - 80 ms RTT, queue length 5 . . . 104
7.10 SBPN2 vs SBPN2 with ring - 80 ms RTT, queue length 32 . . 105
7.11 SBPN2 vs SBPN2 with ring - 80 ms RTT, queue length 32 . . 106
7.12 Ring buffers vs SBPN2 with ring - 80 ms RTT, queue length 5 107
7.13 Ring buffers vs SBPN2 with ring - 80 ms RTT, queue length 5 107
7.14 Ring buffers vs SBPN2 with ring - 80 ms RTT, queue length 32 108
7.15 Ring buffers vs SBPN2 with ring - 80 ms RTT, queue length 32 108
7.16 Ring buffers vs SBPN2 with ring - 150 ms RTT, queue length 32 109
7.17 Ring buffers vs SBPN2 with ring - 150 ms RTT, queue length 32 109
7.18 Ring buffers vs SBPN2 with ring - 150 ms RTT, queue length 5 111

8.1 Unmodified DCCP vs SBPN2 with ring - 80 ms RTT, queue lengths 5 and 32 114
8.2 Unmodified DCCP vs SBPN2 with ring - 80 ms RTT, queue length 32 115

List of Tables

4.1 Skype traffic characteristics 53

4.2 MOS Score and Quality description 60

5.1 Loss and delay with unbounded queue length, 80 ms RTT . . 68

5.2 Loss and delay with queue length 32, 30 ms RTT 84

5.3 Loss and delay with queue length 32, 80 ms RTT 84

5.4 Loss and delay with queue length 32, 150 ms RTT 85

7.1 Loss and delay with queue length 5, 80 ms RTT 99

7.2 Loss and delay with queue length 32, 80 ms RTT 101

7.3 Loss and delay with queue length 5, 80 ms RTT 104

7.4 Loss and delay with queue length 32, 150 ms RTT 110

7.5 Loss and delay with queue length 32, 80 ms RTT 110

Chapter 1

Introduction

1.1 The problem: Increasing media traffic and congestion control

There is an increasing demand for the use of real time media on the Internet and on private networks. Applications such Skype are being used widely on the Internet [72] and these feature video as a component. Demand for these applications will increase as the rate of broadband usage around the world continues to rise [6].

The ongoing debate around net neutrality [83] [49] indicates that there is congestion occurring on the Internet and that ISPs are carrying out traffic shaping because of this. The congestion may be due to operational constraints such as existing network segments running at full capacity, or due to financial constraints such as cost of bandwidth purchased from third parties. Regardless of the cause of the congestion the consumer suffers from a reduced quality experience especially for real time media applications that are affected by congestion more.

The current congestion control mechanisms for the Internet date back primarily to the 1980's and were designed to stop congestion collapse with the TCP over IP (TCP/IP) traffic typical of that era. "Real time" media applications, in their current form, did not exist at that time and the congestion

control mechanisms in place do not apply congestion control consistently for these types of applications as they use mostly User Datagram Protocol (UDP) [94] traffic as outlined in Section 4.4. There is also a risk of congestion collapse due to the increased UDP traffic used by real time media applications and other applications that use UDP.

Quality of Service (QoS) is deployed on network routers and other network equipment to give priority to network streams such as Voice over IP (VoIP). From personal experience, and from discussions with network managers, it is used mostly by medium to large businesses. This prioritisation applies to an entire network stream from a device and not within a stream. However it has been shown that users have a strong preference for audio over video [100] and for having it in a timely manner [58]. It is common in media applications for audio and video data to be interleaved in a single network stream. As such there may be value in prioritising some data within a stream, as well as between streams.

With real time media, if the transmission rate is below the rate that the application requires, then the quality will drop. The lower quality will continue until the rate improves, or the application protocol changes to a lower transmission rate, for example by increasing compression or changing codecs. During this period of lower quality the user experience will suffer due to data not arriving, or not arriving on time. The transmission scheme for real time media should be able to adapt to these situations, and a scheme that did could give significant improvements in quality for real time media. This is particularly applicable when the amount of data being transmitted is being reduced via higher compression, or is already at the codec with the lowest bandwidth requirements. It is appropriate to consider if the traditional methods of data transmission for these applications is still the best method possible.

1.2 The proposed solution

The hypothesis investigated in this thesis is:

Real time media applications will give measurably higher quality audio when there is suitable feedback between the network stack and the applications allowing prioritisation of data within a network.

In a typical application data will be prepared and transferred to the transport layer where it is segmented into packets. A packet is queued and then sent when it reaches the head of the queue, after all other data before it has been transmitted. This however takes no account of time requirements on the delivery of data, or the possibility of different priority for different packets. It is proposed that, by having cross-layer communication around the type and the time constraints for the data, that data can be re-ordered or dropped and that more of the most important data can be transmitted.

This thesis discusses a scheme where the network transport layer, in cooperation with the application, determines which packets should be sent and in what order. The motivation for this is two-fold. It seems reasonable for the networks transport layer to prioritise the sending of the most useful data first, and to discard packets which will no longer be useful for the receiver. In media streams, for example it would also seem reasonable for audio packets to be sent before video packets, as the audio is more important to a user's experience in a real time media application [100]. The application would need to add extra information when the data is placed in the network transport layer queue, but this would not be an odious requirement as this information is known to the application already.

This contrasts with QoS using multiple streams where decisions on dropping packets and which stream gets transmitted first is applied on system or network wide policies, rather than an understanding of the data by the network transport layer in the operating system.

It is proposed that, by using this cross-layer communication around the type of data and the time constraints for the data that the data can be re-

ordered or dropped and that more of the most important data can be transmitted. It is important to deliver the "right" data. With applications it is not sufficient to just deliver data — it is also important to deliver the highest priority data first.

This thesis uses the example of real-time video conferencing to examine this proposal. With video conferencing often audio breaks up under low bandwidth conditions and improvements could be made to this, by increasing the quality of audio at the expense of the video quality.

1.3 Overview of thesis

This thesis investigates the hypothesis and tests the improvements that can be made to real time media by making changes to the transmission of packets. It introduces an algorithm called Send the Best Packet Next (SBPN). This is implemented in the Linux kernel network stack as an extension on top of the Datagram Congestion Control Protocol (DCCP). It is then tested with a video conferencing model that was constructed by analysing existing clients and comparing results to an unmodified transmission scheme.

This chapter states the thesis problem and outlines the contribution expected to advancing audio quality in real time media. In Chapter 2 the use of congestion control and quality adaptation for real time media is discussed. The chapter shows different mechanisms that have been proposed to improve the quality of both stored and real time media, and how quality can be altered by different encoding methods. Congestion control is also introduced with a discussion of the motivation for it's use. As part of this DCCP is introduced. DCCP is the network transport protocol with congestion control used for testing in this thesis. The literature discussing both the interaction of media quality and congestion control is discussed in the last part of Chapter 2.

SBPN is introduced in Chapter 3. SBPN is proposed to improve the timeliness of audio, and to a lesser degree video, that arrives, thus improving the user

experience. SBPN looks at which data to send next on a packet by packet basis considering the priority of the data and the expiry time of that data. With real time media applications it is not sufficient to just deliver more data — it is also important to deliver the highest priority data.

Chapter 4 covers the implementation of the necessary tools and software to test the proposed improvements. The evaluation framework is approached from a practical implementation and testing approach, rather than a theoretical and simulated solution. A model of video conference traffic was constructed by capturing traffic from existing video conference applications. The steps taken to create a test suite are described, and the way that quality improvements are measured is detailed. The work undertaken in the Linux kernel to enable the use of SBPN and the implementation of DCCP in the Linux kernel is then described. DCCP [66] is a new Internet Engineering Task Force (IETF) standards track protocol that has been designed around the needs of streaming media. DCCP implements congestion control and sessions at the transport layer, unlike UDP which does not have these and requires that they are implemented at the application layer.

In Chapter 5 the results from automated tests of quality that were carried out are presented and analysis of the results are given. The quality measures are defined by the International Telecommunication Union (ITU) and are based on how users perceive quality. These tests are carried out with varying network conditions, buffer sizes, Round Trip Time (RTT)s and queue lengths to determine which factors affect the performance of the SBPN1 algorithm when tested compared to unmodified DCCP. The impact of loss is contrasted to the impact of congestion on the SBPN1 algorithm. From the findings SBPN2, which is a refinement of the SBPN1 algorithm, is introduced. SBPN2 is tested and is shown to give substantial improvement in audio quality under constrained network conditions.

Chapter 6 compares the use of Faster Restart and Last In First Out (LIFO) queues to see if these offer improvements to video quality. Faster Restart is a

Request for Comments (RFC) proposal to increase the rate at which DCCP sends packets after loss occurs. LIFO queues are tested to compare to the standard First In First Out (FIFO) queues. Chapter 7 takes ring buffers as proposed by Kohler and Lai [69] and compares these to SBPN and also tests the two combined together to see where further improvements can be made.

The conclusions are in Chapter 8 and ideas for future investigation are also discussed.

1.4 Contribution of this work

The main methods of managing network communication quality at present are QoS and rate changes from codecs. The contribution of this thesis is to show that a significant improvement to quality can also be made by passing information from a real time media application about the priority and expiry time of data to the network transport layer in the operating system, which then can use this information to reorder transmission and selectively drop data.

The work of this thesis depends on an underlying congestion control protocol to provide information on which packet re-ordering and dropping decisions can be based. For practical work DCCP was used for this purpose. Therefore as part of the practical work required for this thesis a substantial body of work was also carried out to help get DCCP code into the Linux kernel and then continuing to improve this in conjunction with researchers at WAND, Luleå, RedHat and the University of Aberdeen. This work included resolving issues with the existing CCID3 implementation and I implemented the draft of Faster Restart algorithm. This practical work was given as feedback to the IETF working group for DCCP to help define future direction.

The thesis hypothesis was investigated by developing, implementing and installing into the Linux kernel a novel algorithm, SBPN, which uses information from the application and the network congestion control protocol (DCCP) to reorder packets and discards packets that are considered no longer useful.

As discussed in Section 1.2 audio is more important to users than video in a video conference. When simulations of video conferences with the SBPN transmission scheme were tested, significantly more audio arrived on time when compared to FIFO transmission giving improved values of standard ITU perceived quality measures. This contribution has significant potential for improving the quality of video conferences and live TV broadcasts over the Internet, particularly when network conditions are variable. There is also potential to extend into other application types such as online games where some data may be more important than other data to be delivered in a timely manner.

Chapter 2

Background

2.1 Introduction

This chapter sets the background for the thesis and outlines related research. The problem of congestion is introduced, and its effects on real time media applications are discussed. The chapter then outlines the two main ways that this has been addressed in the past — through network protocols being improved, and the interaction between media applications and the network protocols. This background prepares the way for our research which combines elements of both to propose a way to improve audio quality in real time media applications.

Section 2.2 introduces congestion. It is discussed in relation to the Internet, and how it is related to real time media requirements. Two differing approaches to dealing with congestion are discussed — using Quality of Service, and dealing with congestion using the end-to-end principle.

In Section 2.3 different network protocols are described, the characteristics of each protocol and how these protocols deal with congestion. The two main transport layer protocols that are used on the Internet are discussed: TCP and UDP, and we also discuss a newer transport layer protocol SCTP, and RTP which is commonly used by media applications. DCCP, the transport layer protocol used in this thesis, is introduced and the rationale for this pro-

tocol being created is explained. The relevance of congestion control to media applications is detailed and how congestion control helps media applications is also discussed. Congestion control can also cause problems for real time media applications as well and this is explained in this section.

Section 2.4 discusses how network protocols interact with media applications. In this section previous work aimed at improving the transmission of video is examined. Differing strategies are examined such as queue management, experimental network protocols, modifications to applications and changes to congestion control mechanisms.

With these themes of congestion, protocols and network protocol interaction the different components affecting real time media are shown. These themes are not connected together and by discussing these it demonstrates the background and paves the way for our research to combine them together to improve real time media applications.

2.2 Congestion

The Internet is not one network but a collection of networks joined together — the word Internet is derived from internetwork or interconnecting networks. The Internet is based upon a layered principle, with layers for application, transport, inter-network and link with data passed between the layers. The operation of the Internet at the inter-network layer is based on data being divided into packets that are transmitted and received using Internet Protocol (IP) as the underlying protocol. An IP packet will traverse a path from the source to destination on the Internet potentially travelling over a number of networks and connections. IP is an unreliable protocol in that it does not guarantee that traffic will be received at the destination. With IP the packets may get discarded, corrupted or delayed and they may not arrive in the order that they were transmitted.

Congestion occurs whenever there is any form of “bottleneck” on a network.

This bottleneck may be due to factors such as links not having enough capacity to handle the network traffic being transmitted and routers not being able to forward traffic at the rate it is arriving. Congestion will result in packets that were transmitted being lost or delayed. With video being over half the traffic on the Internet [8] packet loss and retransmission from video will have a major effect to congestion on the Internet. Packets may also be lost because of factors such as radio interference on wireless links, which may then generate congestion if packets need to be transmitted again.

In practice it can be difficult to tell when congestion occurs unless tools such as Explicit Congestion Notification (ECN) [96] are utilised. At present ECN is not yet widely used [77] on the Internet.

The focus of this thesis is on how real time media applications can respond to the effects of congestion.

2.2.1 Media applications and congestion

The quality of media applications depends on factors such as frame rates, coder-decoder (codec) and picture resolution. The frame rate is how many frames of video are transmitted per second. A codec is an algorithm that converts audio and/or video to a digital representation and often compresses the data. The picture resolution is the number of pixels in a frame of video. These factors determine a bit rate which is the amount of data transmitted per second.

Real time media applications, like other applications, will face congestion where there is any form of bottleneck restricting traffic. This congestion will result in lost or delayed traffic. With real time media, congestion may manifest itself as no sound in an audio stream or distorted sounds occurring. In a video stream with congestion there may be jerkiness (due to the video not being updated), corrupted video or no video at all.

It is therefore our proposition that it is desirable for real time media applications to respond to congestion effectively, both to improve the user experience

and to improve the stability of networks.

From the application's perspective the desired result is that the application adapts to the congestion and improves the user experience. It may be that timeliness is more important than having a perfect picture and sound from the application user's perspective. From a network perspective it is important to reduce the transmission rate during periods of congestion to stop networks becoming overloaded.

Once the application has been notified of congestion it would be desirable that an application adjusts it behaviour, and this forms part of our proposed solution. The application can adjust its behaviour by reducing the quality of the transmission, dropping parts of the transmission or stopping the transmission.

In the history of the Internet the lack of application response to congestion has not proved a major problem as most applications have not had a real time requirement (where packets need to be delivered in a fixed timeframe). For example e-mail is delivered on a queued basis using Simple Mail Transport Protocol (SMTP) and normally arrives quickly but can take a number of hours to arrive. Another example is web traffic which uses Hypertext Transfer Protocol (HTTP) - the emphasis is on all content being delivered rather than the content being delivered in the absolute quickest time. Services such as email and web pages may seem to be real time, but they are normally just delivered very quickly.

Real time traffic needs to meet a deadline. Real time media applications have a different set of requirements; the data must be delivered within a specified timeframe or the data is not worthwhile. This is particularly true for live applications such as video conferencing or Voice over IP (VoIP). Real time traffic will have requirements for minimum bandwidth available, where non-realtime traffic will not.

Video services such as YouTube [14], Netflix [11] and Hulu [10] are also commonly used on the Internet and their congestion control needs are simpler

as they are not real time. With these services all data arrives and if there is congestion that drops the transmission rate below the rate required for viewing, then the video can pause until more data arrives. This is not appropriate for real time media such as a video call between people using technology such as Skype [12] or watching a live sports event.

In summary real time media applications need to be able to respond effectively to congestion to ensure the best quality experience.

2.2.2 Approaches to Ensuring Service Quality

With the Internet individual hosts are responsible for controlling congestion, rather than congestion being centrally managed on the network as often occurs on carrier or corporate networks. The individual end hosts being responsible for network functions is known as the end-to-end (e2e) principle as described by Saltzer [99]. As such the responsibility for congestion control lies within the network transport layer software on the hosts. Different transport protocols take different approaches, as described in Section 2.3.

This contrasts with the Quality of Service (QoS) approach where each and every node that a packet goes through has the same set of rules to follow and each node must uphold these rules. When the rules are updated each node must also be updated which becomes very difficult on a public network such as the Internet. Much of the earlier research on real time media focused on building QoS that gives guaranteed quality and bandwidth for real time media applications [25] [68] [87] [32]. More recently QoS measures such as Differentiated Services (diffserv) [21] have been used, but only within a network or a network provider.

Differentiated Services (diffserv) [21] sets a 6 bit value to classify the type of packet and which indicate how the packets should be treated by the network. These values can indicate, for example, whether traffic should be low loss, low latency or best effort. This can be applied across a network controlled by one organisation but there is no agreement on definitions between companies.

Diffserv also depends on all nodes along a network path implementing it.

As QoS depends on fundamental changes to the structure of the Internet we have chosen to focus instead on ideas that can be implemented on individual devices, using end-to-end principles, as discussed in the following sections.

2.3 Protocols

In Section 2.2.2 it was noted that e2e principles mean that congestion is handled by the individual hosts. The response to congestion is implemented at the transport layer, which is the layer above the network layer (IP). This section discusses the approach taken by different transport layer protocols to congestion.

2.3.1 TCP

Transmission Control Protocol (TCP) [54] is a connection based, reliable protocol that is widely used on networks and the Internet. TCP achieves reliability by requiring that data is acknowledged as being received and can thus ensure that all data is received. If data is not received then it is retransmitted.

2.3.1.1 TCP congestion control

During the 1980's the Internet became unstable in large part due to congestion caused by TCP traffic. Congestion was caused by packets being retransmitted because of acknowledgements not being received. These acknowledgements were often for packets that had already been received, or were still in the process of being transmitted. As queues would build up, the Round Trip Time (RTT) would increase and the retransmission rate would also rise which would further exacerbate the issue and lead to a state that Nagle called "congestion collapse".

Nagle [81] and Van Jacobson [61] proposed a number of changes to TCP to respond to congestion collapse. The changes have become commonly known

as congestion control.

Nagle proposed in 1984 that no new data be sent until an acknowledgement is received for previous data, to prevent spurious retransmission. Nagle also proposed improved use of Internet Control Message Protocol (ICMP) source quench [95]. ICMP source quench packets are sent when a device is running low on buffer space, to request that the sender lower its sending rate (thus reducing the risk of the buffer space being exhausted). The use of ICMP source quench was later deprecated as it was not effective at reducing congestion [44].

Van Jacobson in 1988 proposed that if "conservation of packets" was observed then TCP flows would be generally stable. With conservation of packets when the connection is running without congestion, a packet is only put on the network after an old packet is taken off the network. Conservation of packets was implemented using a congestion window which would be dynamically resized until the connection reached an initial state of stability, and adjusted as conditions changed. The congestion window defined a maximum number of unacknowledged packets that can be in flight at any point in time. Additional packets would not be added when the congestion window was full, until another was removed after receiving an acknowledgement.

Van Jacobson proposed that the congestion window is initially set to one packet and is then increased by one packet per acknowledgement, which has the effect of almost doubling the window size per RTT. This continues until either:

- the maximum window size is reached
- the slow start threshold is reached
- packet loss occurs.

The slow start threshold is an initial congestion window size that is decided by the operating system implementation or based on known network characteristics. If congestion occurs (detected through multiple duplicate acknowledgements or timeout) then the congestion window and slow start threshold

are altered. A more detailed explanation is provided by Stevens [105] and RFC 5681 [16].

These changes by Nagle and Van Jacobson are credited with preventing ongoing TCP collapse [41], [55] and has given stability on the Internet as it has undergone rapid growth.

2.3.1.2 TCP and real time media traffic

With real time media applications, retransmission in the event of a loss or corruption of a packet is usually not the desired outcome as the data is time sensitive and this will add delays to the data. The way most TCP implementations adjust their congestion window is useful for other applications, however for real time media applications it is less suitable due to the sudden change in transmission rates that can occur. With TCP data being sent and received in order a retransmission will cause the receiving operating system to wait for the retransmitted packet before any further data is given to the receiving application. This causes jitter which is a variation in the delay between the sender and the receiver. The receiving application will either have to drop the data with subsequent drop in quality, or have a larger playout buffer to smooth over this jitter. A playout buffer is a buffer at the receiver which holds data for a period of time before it is played, to allow the necessary packets to arrive. Having a larger playout buffer will reduce jitter but will mean extra delay in playing back what has been transmitted - an illustration of the effect of extra delay can be witnessed when comparing the effects on conversation of a local call versus a call to the opposite side of the world. Longer delay makes conversation difficult, as an example, as people are not sure when the other party has finished talking, or they talk over top of the other party.

Retransmitting data uses extra bandwidth with information being sent again. If a packet was dropped due to congestion (as opposed to an error in transmission) then the retransmitted data will probably take the place in the queue of what would be more timely data. If the TCP buffers are large

enough and a queue builds up, potentially the time the data is held in the queue could result in all packets arriving later than required for a real time media application. The timing with real time media is discussed further in Section 4.6.1 and illustrated in Figure 4.5.

The way most TCP implementations adjust their congestion window is useful for other applications but tends to have a detrimental effect for many real time applications. TCP uses an Additive increase/multiplicative decrease (AIMD) [24] approach to changing its rate in that it increases slowly, but decreases rapidly. In many cases TCP will halve the congestion window when a packet is lost.

If the allowed transmission rate is reduced rapidly, it may be necessary to change the quality of the video stream by increasing the compression rate, dropping the resolution, dropping the frame rate or other measures. The allowed transmission rate will then slowly increase, with quality able to increase until network congestion occurs again and then the transmission rate has to be reduced rapidly. This creates an oscillation type effect of alternating between lower and higher quality streams constantly which does not lead to a good user experience.

A datagram based protocol is where all data within a packet is self contained and not reliant on other packets. The TCP protocol is not a datagram based protocol and as such can combine packets and split packets. This can also have a detrimental effect on real time media transmission. If a small packet is queued for transmission then TCP implementations will often wait for a period of time to see if another packet is going to be transmitted so they can combine the later packet/packets so that only one packet is transmitted. Combining the packets increases delay and jitter and this can become particularly problematic with voice packets as these are typically small, and are more time sensitive. In many operating systems this can be turned off on request by setting the MSG_OOB flag. In our testing, confirmed by reading the source code of the Linux TCP stack, this request is not always honoured and packets

are sometimes delayed even if the MSG_OOB flag is set. Splitting of packets can also cause problems if the second or subsequent packet for a particular piece of data is delayed.

2.3.1.3 Improvements to TCP

Development work continues on TCP and congestion control with variations to the TCP protocol such as Vegas [23], Westwood [73] and Binary Increase Congestion control (BIC) [111]. Further details of the advancements in TCP congestion control, particularly in regards to Linux, are outlined in our earlier work [75]. Most of these improvements are based on experimental data but there are also efforts such as Paganini [85] who are approaching TCP congestion control based on provable mathematical modelling.

RFC 4653 [20] proposes to improve TCP in its response to events that are not congestion related, particularly packet reordering. The RFC proposes that the test for congestion should change from three duplicate acknowledgements to a larger number as this would allow more time for out of order packets to arrive.

Floyd introduces quick start [36] that looks at setting up the maximum rate needed through TCP options therefore bypassing the need for slow start, provided that the network path supports the rate requested. For quick start to be implemented, every router that the connection passes through must support it. This is a high barrier for widespread adoption on the Internet.

Goel [18] reduces the TCP send buffer to reduce the latency of TCP packets sent. With a larger send buffer, larger queues can result which increases latency. This is an effective approach but can have the negative effect of reducing send rate, particularly with high RTTs. Goel explicitly targets their work at lower RTT networks.

2.3.1.4 The importance of TCP

TCP is the default protocol for most traffic on the Internet — John and Tafvelin [62] report that TCP traffic is over 90%. As such it is important that any changes to protocols are friendly to TCP traffic flows.

Medina [77] discusses the adoption rate of TCP extensions on Internet, and shows that it takes many years for changes to be adopted across a significant proportion of hosts on the Internet. As changes to the Internet take a long time to occur then the importance of being TCP friendly remains.

2.3.2 UDP

User Datagram Protocol (UDP) [94] is a connectionless, unreliable protocol without congestion control. Originally UDP was used for services such as Domain Name System (DNS) for which just one packet is transmitted and one packet received. With these requirements extra features such as a connection based protocol, congestion control or retry mechanism at the transport layer are not needed.

Real time media applications may use UDP to overcome the limitations of TCP described in the previous section. As UDP is datagram based this overcomes some of the timing problems of TCP for real time media applications. Being datagram based means that, provided the network is able to, data will be transmitted when requested rather than being combined or split. The unreliable delivery of UDP means that it is not guaranteed that packets will reach their destination and they will not be retransmitted if they are lost. The advantage for real time traffic is that in most cases there will be more recent data to transmit, rather than retransmitting stale data. The primary advantage of UDP is therefore that the application developer can choose the timing of packet transmission and how the application responds to network conditions.

A common approach for UDP based media applications is to implement their own form of congestion control. Applications implementing their own

congestion control results in a large amount of duplicated effort or congestion control which is not efficient or unfair to other network traffic. These problems could be overcome by the use of standardised libraries although these do not appear to be in widespread use with media applications. In the worst case scenario a real time media application may use no congestion control at all allowing packets to be discarded and creating congestion in a network, unless networks are provisioned to have a large amount of spare capacity to allow real timed media applications to run at the transmission rate that they require.

Guo [45] has reported that most streaming multimedia traffic uses TCP instead of UDP. Guo states that this is because the lack of a session causes UDP difficulty in traversing many Network Address and Port Translation (NAPT) [104] devices such as home routers or company firewalls. Some media applications are written to use either UDP and TCP and will revert to using TCP if they are unable to use UDP. The use of TCP instead of UDP is becoming a bigger problem as home users switch to broadband connections behind firewalls, utilise NAPT due to limited IPv4 addresses, and extensively use real time media applications. Some applications such as Skype have largely overcome the NAPT problem. Firewalls are also improving support for UDP, and IPv6 with it's larger address space, in the longer term, will remove the need for NAPT. The IETF has provided guidelines for application developers for usage of UDP [30].

2.3.3 RTP

Real-time Transport Protocol (RTP) [102] was designed as a protocol that sits above the transport layer for transferring real time data. RTP is commonly used for media applications on the Internet. It includes the use of a control protocol, Real-time Transport Control Protocol (RTCP) that monitors a range of measures such as jitter, loss and packet delivery. The information gathered by RTCP can be used in real time media applications to understand the underlying network conditions. RTP packets contain information such as the

codec used, timing information and supports the use of layers. RTP is usually implemented on top of UDP but can be implemented on top of other transport layer protocols also. Although RTP is not used in our research, it may be practical to implement aspects of our findings in real time media applications using RTP — especially given its common usage and that the protocol covers both network conditions and the application data.

2.3.4 SCTP

Floyd and Fall [41] discussed in a 1999 paper that the main danger to the stability of the Internet is lack of congestion control causing congestion collapse due to flows that do not reduce their transmission rates when packet drops occur. UDP in particular is discussed, with its lack of congestion control as outlined in the previous section. A number of protocols, such as Stream Control Transmission Protocol (SCTP) [106] and DCCP [66] (discussed in the following section) have been proposed for newer applications such as real time media which aim to overcome the lack of congestion control. Although the Internet has not had widespread congestion collapse there are still risks and Internet Service Provider (ISP)s often use "traffic shaping" to protect their network from heavy traffic and maintain stability. Traffic shaping is where the ISP will deliberately drop packets, either to reduce the amount of traffic being transmitted generally or to reduce traffic of a particular type e.g. UDP or BitTorrent.

SCTP is a protocol that is at Proposed Standard status with the IETF and is a congestion controlled, message and stream based reliable transport.

The similarities that SCTP has with TCP is that it is session based, has congestion control based on TCP congestion control and is a reliable protocol so retransmits lost packets. The congestion control methods, and retransmission means that SCTP will face the same issues as TCP as outlined in Section 2.3.1.2 for real time media traffic.

SCTP shares the characteristic with UDP that it is message based. This

means that as soon as data can be sent, it will be sent, unlike TCP which may wait to combine packets if needed. If a packet is too large then SCTP will split the packet (like TCP, and unlike UDP) but will not combine the data with other packets. SCTP also shares characteristics with UDP in that it allows out of order delivery, removing the need to wait for a retransmission before any further data can be received which is beneficial for real time media. It is stream based, which means that audio and video can be separated into different streams if desired.

2.3.5 DCCP

Datagram Congestion Control Protocol (DCCP) [65] is a transport protocol that moved to Proposed Standard status with the IETF in 2006. DCCP is a session based, unreliable protocol with congestion control. This means that it is session based like TCP, but unreliable like UDP. The rationale behind the new protocol is that existing protocols do not handle the requirements of modern applications such as multimedia as well as desired. The use of UDP does not provide congestion control at the transport level, and is not session based so in the past had difficulty traversing some firewalls or NAPT devices. More recently some applications have overcome these limitations. As discussed earlier TCP does provide congestion control and is session based but is not as suitable for real time media applications due to retransmission, and the use of AIMD based congestion response (which alters the transmission rate rapidly rather than smoothly). DCCP aims to provide a solution to these problems.

The DCCP protocol is defined in a modular method - there is a base protocol defined, and multiple congestion control mechanisms can be implemented. The Congestion Control Identifier (CCID) specifies which mechanism is used. This allows for DCCP to be extended in the future with additional congestion control mechanisms. At the time of writing there are two CCIDs at Proposed Standard status and one at Experimental status with the IETF:

- CCID2 [38] is TCP-like congestion control and implements a congestion

control mechanism based on TCP. It follows many of the TCP semantics although it does not have retransmission.

- CCID3 [40] is based on TCP Friendly Rate Control (TFRC) [47]. TFRC aims to provide a smoother response to congestion than TCP while still using a "fair" share of bandwidth compared to other flows. TFRC achieves this goal by estimating the sending rate available rather than halving the window in response to congestion as TCP can do.

- CCID4 [39] is experimental and is based on CCID3. It is designed for applications with small, frequent packets such as VoIP and enforces a minimum period between packets of 10 milliseconds.

Phelan discusses the use of DCCP in his DCCP user guide [92]. The user guide recommends the use of CCID3 [40] for multimedia applications although Phelan recognises issues around slow-start and bandwidth decreasing after idle. We use CCID3 for our experimental work in this thesis.

De Marco, Longo and Postiglione [29] build on DCCP with a protocol they label Run-Time adjusted Window-based DCCP (RTW-DCCP) which allows applications to transmit at a rate based on an average throughput. The average throughput is adjusted based on observing changes in the RTT under the assumption that RTT increases are largely a consequence of increasing queues and congestion. RTW-DCCP allows a sending rate that adjusts more slowly than the congestion control of TCP, which rapidly adjusts the number of packets that may be sent. Having congestion control that maintains a sending rate that does not change quickly works well with media applications that transmit at a constant rate, or take time to adjust.

In Balan [31] VoIP quality is compared between UDP, TCP and DCCP. UDP gave the best quality with high loss rates, followed by TCP and the lowest quality was DCCP. TCP quality was worse than UDP because of the retransmissions required which lead to delays in receiving packets. They proposed that DCCP quality was lower than the others because the congestion control

was more simplistic than TCP congestion control. This was tested by adding measures such as quick start and faster restart to DCCP which increase initial transmission speed and recovery from loss respectively. With these changes DCCP quality increased significantly under lossy conditions.

Phelan discusses in an Internet Draft a CCID for DCCP called Media Friendly Rate Control (MFRC) [91]. Phelan recognises that media applications can change their rate quickly, including between frames and will often have periods of silence where network activity can drop to almost zero. CCID2 and CCID3 respond to congestion by using slow start and will also drop the allowed transmission rates in idle periods. MFRC proposes that applications can immediately start transmitting at their desired rate (provided it is within a pre-defined limit) and that applications can quickly return to a proven transmission rate provided congestion is not experienced.

DCCP was developed with one of the aims being to be a better congestion control protocol for media applications. This thesis introduces a new extension, Send the Best Packet Next (SBPN), with the aim to make it easy for media applications to take advantage of the features of DCCP. As such we provide an API to enable media applications to pass information to SBPN and SBPN uses the features of DCCP based on this information.

2.4 Network protocol interaction with media applications

The previous section discussed how network protocols have been adapted to cater for media applications. In this section the interaction between the applications and network protocols is examined.

Adapting media applications to cope with the variability of the Internet has been an ongoing problem and there have been different approaches to solving these issues. Many of these have been through looking at the use of layers and this is further examined in this section. Layers are different parts of a media

stream such as audio and different resolutions of video. These layers may have differing level of importance to the user e.g. audio is more important than video.

de Cuetos and Ross [28] present what they call a "unified optimisation framework" where they combine scheduling, Forward Error Correction (FEC) and error concealment to improve the performance of video transmission in lossy environments. With FEC extra data is transmitted and if there is an error it may be possible to reconstruct the missing data. Error concealment is a technique where information is gathered from other sources such as preceding frames to replace missing data, mitigating some of the effects of the lost or corrupt data.

In Section 2.3.1.2 the major issue that is faced by video codecs when using standard AIMD based congestion control responses such as standard TCP was discussed. This issue is that the bandwidth available to the application may fluctuate substantially, where real time media applications need a more constant rate. Dai and Loguinov [27] examine this issue and compare congestion controls protocol based on Kelly's proportional-fairness framework [63] with standard TCP. They found that by using congestion control algorithms that change the rate more smoothly than TCP that the quality of real time media improved. This matches the findings of De Marco [29] as discussed in Section 2.3.5 Our choice of congestion control mechanism for SBPN reflects this.

Wang, Banerjee and Jamin [110] examine whether a protocol being TCP friendly means that it is also media friendly. TFRC is used in their tests and they conclude that it is not media friendly as less than the fair bandwidth is used and large variations in rates occur even in steady state networks. By fair bandwidth they mean that the same bandwidth is obtainable as with TCP. Phelan [90] also discusses the limitations of TFRC and how real time media applications can adapt to the limitations and variations that are proposed for VoIP for TFRC. He suggests sending test packets to find the bandwidth

available and dynamically changing codecs used if the bandwidth available changes over time.

Bolot and Turletti [22] put in place a feedback loop to measure the data rate achievable over the Internet (or private network) without using QoS and then adjust the output bit rate to the achievable rate calculated from this measurement. They decreased the bit rate by decreasing the frame rate, changing movement detection thresholds and decreasing frame quality through changes to the quantizer. Movement detection reduces the amount of data transmitted by using similarities between frames and quantization is the process of mapping a large range of values to a smaller range of values, for example reducing the number of colours used in a frame. With these changes they were able to improve the quality of video.

In summary, media applications try to understand what is happening in the network and see how this can give improvements to the application layer. This can be through protecting the data through the use of FEC to transmit extra data to recover from errors and using error concealment to mask any data that is lost. Media applications are better suited to using different congestion control then AIMD schemes and TFRC has been shown to be better as the response to congestion is more gradual. Applications can also measure the network bandwidth available to them and then adapt what is transmitted to adjust for link capacity and quality dynamically.

2.4.1 The use of audio and video layers with congestion

In this section we look at the use of layers for audio and video transmission, and their interaction with congestion control. This makes up a core part of how our solution works, particularly in prioritising which layers get transmitted.

Rejaie, Handley and Estrin [98] and Feamster [33] propose that video codecs should add and remove video layers as network conditions permit. Rejaie, Handley and Estrin [98] further the earlier work of Bolot and Turletti by adjusting the bit rate through adding and removing layers of detail for video

and audio transmission. In their approach they smooth the transitions between layers so that the buffers are not drained or overflow. As part of this they average the bandwidth available and take a conservative approach to adding extra layers to the transmission. This work is further developed in Rejaie and Reibman [97] where the transport and the encoder become "aware" of each other to enable decisions to be made in regards to changing the number of layers transmitted, and adjusting for loss events.

In Gharavi [43] a similar scheme is proposed but the transmission rate is calculated by measuring the number of hops between the source and destination and adjusting the number of frames per second with more hops resulting in a lower transmission rate. If the number of hops increases or decreases during the session then the transmission rate is decreased or increased respectively, as their research showed that the number of hops changing during a session indicated that the network was altering routes due to factors such as failure and congestion. Their research showed that this reduced the loss rate of packets when compared with making no change as the number of hops changed.

In a study of multimedia streaming Guo [45] looked at the time taken to drop transmission speed to a lower bit-rate in response to a reduction in the available bandwidth. Guo found that the median time for a codec to change to a lower bit-rate stream was around 4 seconds. This length of time indicates that there is scope to improve the user experience during the transition.

Feng [34] builds upon changing the selection of video layers and proposes a priority queue for delivery of pre-recorded video streams. In this system the minimum needed to create a low quality video stream at the desired frame rate is sent first. Enhanced quality is only sent after the high priority video layer has been sent, and if there is network capacity available. This ensures that the frame rate is maintained and the picture quality improves as the network conditions improve. Krasic [67] investigated storing streaming video in multiple layers tailored to the type of client requesting the data. The streaming server, which they call priority-progress, then decides the most appropriate layers for

the client and maps these to a priority order within each time segment. At the end of each time segment, if the data has not been transmitted, then the data is discarded by a "progress regulator".

In Ahmed, Mehaoua, Boutaba and Iraqi [15], Moving Pictures Expert Group (MPEG)-4 video transmission is adjusted for congestion based on prioritising the objects that are sent. Highest priority is given to the audio part of the stream and lowest to a logo on the screen. They encode the packets with markings for diffserv and the routers drop packets as required.

Papadimitriou introduces Scalable Streaming Video Protocol (SSVP) [86] and Video Transport Protocol (VTP) [19]. These are both built on top of UDP. SSVP adjusts the send rate through altering the inter-packet gap, VTP smooths the transmit rate and sends at a fixed rate. Both protocols adaptively alter the layers of video being transmitted. Amer [17] introduce a partial order transport service that allows reliable or unreliable partial order service. This allows marking of importance and time values by the application. This is used by the receiver to determine whether lost data is ignored or requested for retransmission depending on their temporal value.

In Tsaoussidis [107] Multimedia Transmission Protocol (MTP) is introduced. It is based on TCP Reno but without guaranteed reliability. Packet priority information is sent as a 2 bit field which determines whether packets may be retransmitted or not. This enables MTP to not retransmit data which is of lower priority, as compared to TCP which retransmits all missing data. Time-lined TCP (TLCTCP) [80] introduces the concept of tracking expiry time for data. TLCTCP marks data with an expiry time, and discards data if the expiry time is reached before the packet is sent. TLCTCP does not take the RTT into consideration but proposes that it is worthwhile to do so. TLCTCP is a partially reliable protocol - if the data has not expired and retransmission is requested by the receiver then the data is retransmitted. If the data has expired, a more recent packet is sent instead as the expired packet no longer has value.

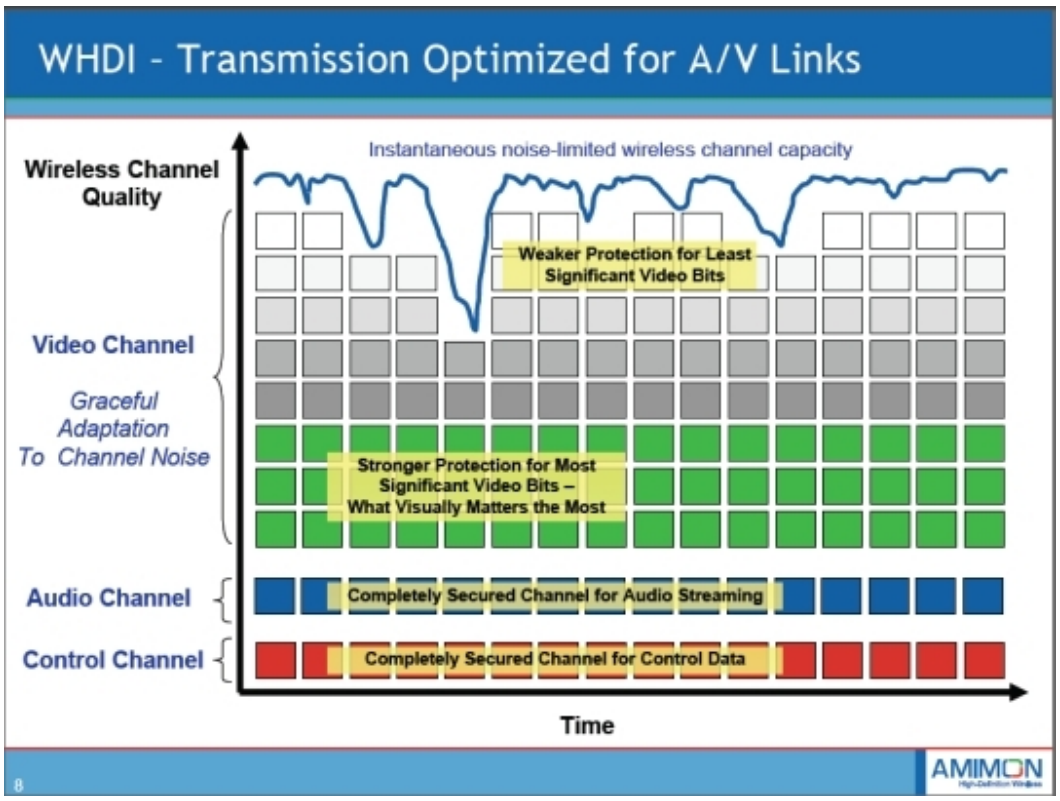


Figure 2.1: Diagram from Ars Technica [51] showing WHDI protecting data in higher priority layers

As a practical example of the importance of layers, Wireless Home Digital Interface (WHDI) [3], which is used for linking home multimedia devices, protects the data in higher priority layers more than lower priority layers as shown in Figure 2.1. In WHDI, control and audio data is completely protected and video data is given varying degrees of protection depending on the importance. The specification of WHDI has not been published, but it is likely that fully protected data relies on retransmission of lost data and that varying degrees of protection are achieved by differing degrees of FEC.

2.4.2 Transmission buffers

Packets are buffered before they are transmitted. These buffers are examined in our research to see if they can be altered to improve real time media performance. In a FIFO queue the data is transmitted in the order that it was received. TCP and UDP implementations typically use FIFO queues as pack-

ets are transmitted in the order received from the application. A LIFO queue works on the principle that the last packet received into a queue is the first transmitted. FIFO and LIFO queues can be fixed or variable length.

In a ring buffer packets will get overwritten if they are not transmitted. The term ring buffer is used as the start and the end of the buffer can move and it can be considered to be circular in nature. If the buffer is full then the next packet inserted overwrites the oldest packet in the buffer which is discarded. This contrasts to FIFO and LIFO queues where the newest packet is discarded if the queue is full.

Research by Lai and Kohler [69] investigated using late binding ring buffers with DCCP for transmission. The ring buffers were writeable directly by the application, where with most transmission schemes the buffers are controlled by the operating system. The application could replace a packet in the buffer with a more recent packet if it had not been transmitted. These buffers are called late binding as the next packet to be sent can be changed up until the time for transmission. The aim is for the most timely data to be sent and old data to be discarded. They found that using late binding ring buffers meant that more of the important video frames were received.

One of the aims of our solution is to allow the flexibility of ring buffers to improve the real time media experience but not to burden the application programmer with the need to work out which packets to drop next if the network link is congested.

2.5 Summary

This chapter has discussed congestion in Section 2.2 and measures taken to stop congestion collapse on the Internet. Congestion control has an effect on real time media applications and there is scope for improving the user experience on congested networks, particularly as network conditions change.

Different protocols such as UDP and TCP were discussed in Section 2.3

and their limitations in regards to real time media applications were discussed. TCP has congestion control and this has continued to develop, but does not take into account the characteristics required of real time media applications. UDP is often used for real time media applications but is missing components such as connections and built in congestion control. The use of RTP as a layer above the transport protocol was explained and a stream based protocol, SCTP, was also detailed. The background for DCCP was explained and the different parts of the protocol detailed. DCCP is the protocol that is used as the basis for our solution as it is an unreliable protocol that has sessions, and the congestion control is suited for real time media applications.

In Section 2.4 Quality of Service (QoS) is explained and it is explained that it is not used in our research as it needs implementing on all nodes that encounter congestion and we are looking at solutions that can be deployed on the Internet by implementation only at end points. The research into the interaction between congestion control and layers within a video stream was also detailed in this section. The section ended with different forms of transmission queues being explained, including ring buffers which were designed for use with real time media applications.

The previous work shows that improvements can be made in applications, that improvements can be made in network protocols and improvements can be made in congestion control but this past research has largely not covered the interaction between these all areas. In particular if a media friendly network transport protocol shares information with media applications we believe potentially significant improvements in quality for real time media applications are possible.

We plan to show that by taking account of the information available between all of these areas a significant improvement can be made to the quality of the user experience. A media friendly network transport protocol could make decisions at the time of transmission about which is the most appropriate data to transmit, without requiring real time media applications to have extensive

modifications, which contrasts to existing layer based approaches which require larger modifications.

In the following chapters we introduce how we propose to make improvements in this area with SBPN and then test these ideas experimentally.

Chapter 3

Proposed solution

3.1 Introduction

This chapter proposes a method to improve the quality of audio in real time media transmission: Send the Best Packet Next (SBPN). The background decisions around the choice of transport protocol are given and details of the first two versions of SBPN are given.

Section 3.2 details the reason for the choice of DCCP for SBPN and describes TFRC, which is used for congestion control in DCCP and the SBPN implementation. In this section the reason that TFRC is more suitable for media applications is outlined.

Section 3.3 outlines the concepts of SBPN. It describes how improvements can be made to the quality of real time media by passing information between applications and the operating system and how this can be used to improve the transmission of packets for media applications.

SBPN1 — the first version of SBPN is discussed in Section 3.4 and the algorithm is described. The algorithm is then explained in depth: detailing which packets are sent, which are dropped and the order of transmission. A development of the algorithm, SBPN2, is discussed in Section 3.5.

3.2 Use of DCCP

3.2.1 Rationale

Datagram Congestion Control Protocol (DCCP) was investigated and chosen for the implementation of SBPN as it is a session-based unreliable transport protocol for datagrams. The IETF required that all new protocols had to have congestion control and the criteria for congestion control was outlined in RFC 5033 by Floyd [35]. DCCP was created by the IETF as there was recognition that the two main existing transport protocols, TCP and UDP, did not meet the needs of newer applications being created. DCCP has been designed to give sessions and congestion control for these applications, such as multimedia streaming, VoIP and online gaming that may not always need complete reliability.

At the time that research was commenced on this thesis DCCP was the only standardised and implemented option. DCCP and its related CCIDs had been through IETF working groups and RFC process, and they are now at Proposed Standard status. Further background on DCCP and its usage for multimedia applications is contained in the Problem Statement for DCCP, [37] a copy of which is in Appendix A, and in Section 2.3.5.

UDP was not used because it does not track the RTT and does not have congestion control in the base protocol, and to implement this at the application level is extra work. UDP also has difficulty traversing some firewalls due to its lack of session control and often applications fall back to TCP.

It was also decided not to use TCP as it is a reliable protocol and if data is not received it is retransmitted when, for real time media applications, it would be better to discard the data as the data is no longer relevant. TCP also combines packets, even when the Nagle algorithm [81] is turned off and this causes delay in packets being sent as TCP waits for further packets to be ready for transmission. The delay causes quality to be lower as described in Section 4.6.2.

DCCP has multiple algorithms for congestion control, each of which is identified by a Congestion Control Identifier (CCID). CCID3 is used for SBPN as the authors of it believed that it was the most suitable for multimedia applications (as outlined in Section 2.3.5). CCID3 is based on TFRC which is outlined in the following subsection.

The choice of DCCP to be the underlying transport protocol was based on practical considerations as well. DCCP was an active research project in the WAND network research group at the time practical work commenced. It was also helpful that DCCP as a protocol measures the RTT and is a datagram based protocol and these features have been used in SBPN.

3.2.2 TFRC congestion control

TCP Friendly Rate Control (TFRC) aims to achieve the same throughput as TCP under the same network conditions while achieving less variation in sending rate than TCP to cater for the needs of multimedia applications. The way that this is achieved is to use the TCP throughput equation as the basis for the sending rate allowed, but to respond to congestion in a different manner than TCP.

The formula for maximum TCP throughput was derived by Padhye [84] and is used in the TFRC specification [47]. This formula is shown in Equation 3.1 where X_calc is the calculated transmit rate in bytes per second, s is the packet size in bytes, R is the Round Trip Time (RTT) in seconds, p is the loss event rate as a decimal fraction, t_RTO is the TCP retransmission time-out in seconds, b is the number of packets acknowledged by one TCP acknowledgement.

$$X_calc = \frac{s}{R\sqrt{\frac{2bp}{3}} + (t_RTO(3\sqrt{\frac{3bp}{8}}p(1+32p^2)))} \tag{3.1}$$

TFRC makes the assumption that $t_RTO = 4R$ and that b is 1. This assumption simplifies the throughput equation to Equation 3.2. TFRC mea-

sures the loss rate, p, and the RTT, R, and uses the equation to determine a maximum transmission rate.

$$X_calc = \frac{s}{R(\sqrt{\frac{2p}{3}} + 12\sqrt{\frac{3p}{8}}p(1 + 32p^2))} \tag{3.2}$$

With TFRC, packets are transmitted at the rate calculated in Equation 3.2 by scheduling time between packets and therefore the transmission rate is altered depending on the network conditions. This scheduling of packets contrasts to TCP congestion control where many packets can be sent at once, followed by a larger time period before the next transmission.

TFRC is designed to address networks which face congestion due to bandwidth constraints which result in packets being delayed or dropped. TFRC is not suitable for lossy networks such as mobile communication where IP packets are lost due to causes such as radio links that are subject to interference and corruption, and the link layer is unreliable. In a lossy network, techniques such as FEC and not decreasing transmit rate upon loss are employed instead to cater for the changes in the network that occur. As SBPN uses TFRC the same constraints apply to the suitability of the algorithm to different network conditions. It is assumed that loss is due to congestion in this thesis.

3.3 Sending the Best Packet Next

In conventional network applications all packets are transmitted by the network transport layer in the operating system in the order that they are received from the application. Packets are discarded if buffers are full, or transmission of data falls further and further behind depending on the transport protocol used.

Send the Best Packet Next (SBPN) is implemented on the end nodes, rather than in the network core or network switches. The primary reason for implementing congestion control on the end nodes, rather than across the network is the ability to implement SBPN without requiring a change on every

hop of a network connection. This is in line with the e2e principle as discussed in Section 2.2.2.

The use of e2e principles also means that QoS measures within the network such as diffserv [21] are not used in SBPN. It would however still be possible to use SBPN in conjunction with QoS measures such as diffserv. The use of diffserv in conjunction with SBPN is a possible area of future research.

Part of the motivation for the writing of SBPN was to allow application writers to have a congestion control mechanism for real time media applications without needing to write their own congestion control as is required when using UDP. With SBPN being based on DCCP, congestion control is provided by DCCP at the transport layer, and then SBPN allows the most relevant traffic to be transmitted first. Other approaches such as ring buffers have been used for media applications as discussed in Section 2.4.2. These allow applications to provide information to the transport layer. With a ring buffer the application decides which is the most relevant packet to be transmitted, where SBPN removes this requirement from the application writer.

SBPN is also completely sender based as it alters the transmission policy only. SBPN does not require the transmitter to send any new data that is not already transmitted, as the RTT and allowed transmit rate are already calculated by DCCP. As such, the algorithm can be implemented on a sender without requiring changes to the receiver. This also has the benefit that improvements to the algorithm could be made on the sending device without changes being required on all receiving devices.

SBPN is based around three main principles to improve performance for real time media applications:

- All packets are not created equal
- Allow applications to pass information to the network transport layer without much overhead on the application
- Allow the network transport layer to alter packet send order and discard

policy

These principles are expanded in the following sub-sections.

3.3.1 Not all packets are created equal

As discussed in Section 2.4.1 multimedia traffic is encoded in layers that consist of control data, audio and video. The control data, that maintains the connection, is typically transmitted using a reliable protocol such as TCP to ensure that no data is lost.

Video is typically comprised of Intra-coded Picture (I), Predicted Picture (P) and Bi-predictive Picture (B) frames. I frames are video frames that can be displayed independently of data from other video frames. P frames use data from previous frames and B frames can use data from both previous and future frames. I frames are sent less often then P or B frames as they contain all the data needed to display a frame, and as such contain more data and therefore use more bandwidth.

If a frame is not received then there will be visual artefacts that will get progressively worse until the next I frame is received. The effect will be even more noticeable if an I frame is not received as the P and B frames will have differential data against a frame that does not exist at the receiver. I frames should therefore be given higher priority than P and B frames to avoid these visual artefacts.

For the purposes of simplicity the experiments in this thesis look at two layers only — audio packets and video packets. Audio packets are considered to be the most important to the user experience, with video packets being lower priority. Control packets are ignored, due to their low volume, and that they are also typically transmitted using TCP to guarantee delivery. Multiple video layers are also not investigated as the core question in this thesis is about improving the audio experience for the user. The research could be extended to multiple video layers in future with I frames being more important for example. In our implementation, as described in Chapter 4, the structure used

in Figure 4.3 accommodates the use of multiple layers as does the underlying code implementation and this can be an avenue for future research.

3.3.2 Passing information to the operating system

If the transport layer is to make decisions on which packets to send next there needs to be a mechanism to pass this information between the application and the transport layer in the operating system using an Application Programming Interface (API). SBPN uses two pieces of information that an application attaches to the packet data — the packet expiry time, and the priority level of the data that corresponds to which layer (e.g. audio or video) the data is from. The packet expiry time is described further in Section 4.6.1.

These two attributes, packet expiry time and priority level, are added to the packet transmission API for SBPN. This potentially allows any real time media application to be altered to allow it to use SBPN and to benefit from improvements in audio quality.

3.3.3 Allow altered packet sending and discard order

If the transport layer has information such as the priority and packet expiry time then the application does not have to actively manage the network transmit queues as is proposed in other research such as Lai and Kohler [69].

Transport layers traditionally transmit packets in the order that they are received. There is scope here for the transport layer to alter the transmission order of packets as not all data is of equal importance for media applications, and more important data can be transmitted first.

As the transport layer in the operating system is aware of the packet expiry time under SBPN the transport layer can autonomously decide whether to discard a packet or not based on a set of rules. This gives the benefit of limited network capacity not being used by packets that will be discarded by the receiver as no longer useful. It is better for the sender to not send the packet and more relevant data to be sent instead.

```
receive_time = current_time + (RTT / 2)
if receive_time < expiry_time then
  transmit packet
else
  discard packet
```

Figure 3.1: The expiry algorithm for SBPN1

3.4 SBPN1 algorithm

To test the premise that SBPN will improve audio quality for real-time media applications the SBPN1 algorithm was created to enable experiments to be carried out.

The SBPN1 algorithm sorts the transmit queue in the transport layer as a priority queue. Audio packets are marked as higher priority than video packets when the application puts them into the transmit queue. As such audio packets will be put into the queue in front of any video packets and present at that time and therefore audio packets will be sent before current video packets.

The packets in the transmit queue are also marked with an expiry time by the application. When a packet is due to be sent by the transport layer, the expiry time is checked against the sum of the estimated time taken to reach the receiver and the current time. If the expiry time is less than this value the packet is discarded rather than being sent.

The time taken to reach the receiver from the time of transmission is taken to be half of the RTT based on the assumption that the link is symmetrical. If the link were asymmetrical, and that could be measured, a more precise measurement could be used to calculate if the packet would expire while in transit.

The pseudo-code for packet discard is shown in Figure 3.1.

Just two priorities are used in SBPN1 - audio and video. This is done for the purposes of simplicity, but the algorithm could be used for multiple priority levels within video e.g. a higher priority for I frames than for B and P frames.

3.5 SBPN2

The results from testing SBPN1 are discussed in Chapter 5 and based on these results it can be seen that further improvements are needed to SBPN1.

One aspect of CCID3 is that it reduces the allowed transmit rate after short idle periods. The formula for allowed sending rate, X, is shown in Equation 3.3 where X_calc is the theoretical TCP throughput equation calculated using Equation 3.1 from Padhye [84], X_recv is the receive rate from the previous feedback (at least once per RTT), s is the average packet size and t_mbi is 64 seconds based on the maximum time between packet transmissions where there is no feedback. In most circumstances the calculation becomes Equation 3.4 as s/t_mbi is approximately 8 bytes per second. This means that if the sending rate is reduced below the level allowed, the receive rate will be reduced shortly afterwards, which may mean that the next period sending rate may be further reduced. This issue has been raised at IETF 68 [4] when discussing DCCP implementation experiences. There is also an Internet draft for TFRC Faster Restart [64] that is discussed further in Chapter 6.

$$X = max(min(X_calc, 2 * X_recv), s/t_mbi) \tag{3.3}$$

$$X = min(X_calc, 2 * X_recv) \tag{3.4}$$

With our transmission scheme in SBPN1 discarding packets when they are expired this exact scenario of the allowed sending rate reducing can, and does, occur. In effect the SBPN1 algorithm is being penalised for sending less than the allowable limit. To address the issue of reduced transmit rate the SBPN1 algorithm was modified to send an expired packet if it is the only packet in the queue to reduce the likelihood of the transmit rate being reduced. This is called SBPN2.

The algorithm for inserting a packet into the queue for transmission with SBPN2 remains the same as that of SBPN1 as described in the previous sec-

```
receive_time = current_time + (RTT / 2)
if receive_time < expiry_time then
  transmit packet
else
  if queue_length = 1 then
    transmit packet
  else
    discard packet
```

Figure 3.2: The expiry algorithm for SBPN2

tion. SBPN1 checks each packet at the time of transmission to see if the packet has expired, and discards the packet if it has expired. SBPN1 can therefore discard all packets in the transmission queue. The algorithm for SBPN2 changes this so that if there is only one packet left in the transmission queue it is sent regardless of the expiry status.

The pseudo-code for this is shown in Figure 3.2.

3.6 Summary

In this chapter SBPN was introduced, and it is proposed that by media applications providing information to the transport layer that improvements can be made to audio quality. With SBPN the traditional method of all packets being transmitted in order by the transport layer is changed so that intelligent decisions are made based on the content of the data. It would seem logical that there is no point transmitting data that will not be used by the receiver, and to transmit audio before video. These two assumptions form the basis for the SBPN algorithm. The first version of the algorithm, SBPN1, was explained in this chapter. A second version of the algorithm, SBPN2, was also introduced to improve on SBPN1.

DCCP was chosen as an underlying transport protocol for use in SBPN as it provides session based unreliable transport that is suited to media applications. TFRC was used as the congestion control mechanism as it provides a throughput that is fair to TCP but changes transmission rate more smoothly

than TCP. TFRC transmits packets on a time based schedule derived from the allowable transmission rate, as compared to TCP which sends packets in groups, and alters the packet sending rate rapidly. This makes TFRC a better choice for media applications that need a transmission rate with less variability.

SBPN is a unique contribution, combining a priority queue, with expiry times for packets that take RTT into account. In the following chapters SBPN will be tested by running experiments and examining the results to determine if improvements are made through the use of the algorithm.

Chapter 4

Implementation

4.1 Introduction

In this chapter the implementation of DCCP and SBPN is discussed; a video conference model is built; a description of the test framework is given and a method to analyse the results that are obtained from testing is proposed.

The implementation of DCCP in the Linux kernel is detailed in Section 4.2 and the history of the DCCP source code is outlined. Technical difficulties encountered, and how those challenges were overcome are described. There is also a description of tools that were used to assist with the DCCP implementation

Section 4.3 describes how SBPN is implemented on top of DCCP in the Linux kernel. An API is described for user programs to interact with the SBPN implementation.

In Section 4.4 three applications — Skype [12], Ekiga [9] and MSN Messenger [5] — are tested to understand the traffic patterns that they use for real time video conferencing. From this a model is produced to use in our tests.

The test framework that is used is described in Section 4.5. The use of Netem [50] is outlined in Section 4.5.1 and the test applications that were created are detailed in Section 4.5.2.

In Section 4.6 the two methods that will be used to measure whether SBPN

makes an improvement are described. A measure "On time arrival" is created and the use of Mean Opinion Score (MOS) is described in detail.

4.2 Implementation of DCCP

It was decided to use DCCP in a Linux kernel, rather than as a simulation in a tool such as ns2 [1]. The primary reason that implementation was chosen to be done in the Linux kernel, was that it was desired to test with real application data. The tools for Linux were available and it was believed that it would be relatively easy to modify the existing DCCP implementation for Linux to meet the requirements needed to carry out testing. In the end, substantial work was required to get a working implementation of DCCP.

4.2.1 History

For the Linux 2.4.x kernel there were two main early releases of DCCP. There was an implementation by ICSI Center for Internet Research (ICIR) [53] to test the difficulty of implementing the specification, and an implementation by Patrick McManus [76] which implemented the base protocol and CCID2. The University of Waikato WAND Network Research Group took the implementation from Patrick McManus and incorporated CCID3 code from Luleå University of Technology [71] which was originally built on top of FreeBSD. The Luleå code was re-licensed under the GPL to ensure that it could be merged into the Linux kernel. This code was tidied for release by WAND and is available for the Linux 2.4.27 kernel [109].

I migrated the WAND implementation to the Linux 2.6.11.4 kernel and at around the same time Arnaldo Carvalho de Melo (a founder of Connectiva Linux) independently started implementing a version of DCCP for the Linux 2.6.12 kernel. Arnaldo's code initially consisted of the base protocol without any implementation of CCIDs. I worked with Arnaldo to merge these two — the main part of the University of Waikato code that was merged was for

CCID3, as well as extensive help in fixing general DCCP bugs, adding code to ensure RFC compliance and testing. This combined code base was accepted into the official Linux kernel tree by Linus Torvalds and was released in Linux 2.6.14 kernel. Improvements and additions to the code continue to be made in the official Linux kernel tree after this initial acceptance.

DCCP was implemented for both IPv4 and IPv6. The code for DCCP was implemented in a modular way to enable the code to be extended in future. The directory structure for the DCCP code in the Linux kernel is:

```
net/dccp            base protocol
net/dccp/ccids      CCID specific
net/dccp/ccids/lib  CCID libraries
```

Further details on the implementation of DCCP can be found in Melo [79].

The version of Linux kernel used for most of the research in this thesis was Linux 2.6.20 kernel with a series of patches [74] to ensure that the performance of CCID3 matched the TFRC [47] throughput equation. Later on, Linux 2.6.23 kernel was used as described in Chapter 6 for TFRC Faster Restart [64] as this was the version on which it was implemented.

When Melo started on his implementation of DCCP he realised that there were many similarities between the current TCP code and what would be required for DCCP. The code was restructured so that much of it could be used for both TCP and DCCP, thus avoiding duplication. In particular TCP sockets and code using these sockets were examined to see if they could be used in multiple protocols rather than just TCP. These changes are a continuation of Melo's earlier work [78]. In the future there is the potential to rework existing protocols such as SCTP to use the restructured code, to extend the code shared between TCP and DCCP, and also to implement other protocols such as XCP. A similar process has also been commenced by Melo for DCCP over IPv6 as well.

As both TCP and DCCP codebases in Linux use modular congestion control mechanisms there is also scope to consider whether the code can be shared

down to the individual congestion control mechanisms for both protocols. This would be particularly applicable for CCID2 which uses TCP congestion control. A shared codebase could also be used to compare real time media performance between TCP and DCCP if the TFRC code were implemented as a shared congestion control mechanism for TCP and DCCP.

4.2.2 DCCP Tools

As part of the Linux implementation of DCCP work was done on a number of supporting tools to help resolve bugs in the implementation.

DCCP support was added to tcpdump [7] in conjunction with work done by Melo. Tcpdump is software used to analyse traffic flows on a per packet basis.

The programs ttcp and iperf [56] were modified to enable use of DCCP and selection of TCP congestion control at run time. These two programs are used to generate traffic and were used to measure throughput speed over varying loss and congestion on a connection to ensure that they matched the specification of DCCP and CCIDs.

A Linux kernel module, DCCP Probe, was also written which allowed the internal state in the DCCP Linux kernel module to be exposed for debugging purposes. DCCP Probe was based on the source code of TCP Probe [2] written by Stephen Hemminger.

4.2.3 Implementation challenges

One of the challenges of implementing the Luleå CCID3 code in Linux 2.6.x was implementing the mathematical calculation of the transmission rate. The challenge was two fold — the lack of 64 bit integer operations on 32 bit architecture and the inability to use floating point instructions in the kernel. This was resolved by converting the Luleå floating point lookup tables to integer based lookup tables with some reasonably complex manipulations. As part of this a large amount of testing was carried out which showed that previ-

ous DCCP implementations based on the the Luleå code had implemented the CCID3 rate calculation incorrectly. Another challenge in implementation was with implementation of locking around packets and this consumed a large amount of time in implementation.

Tests have been conducted between Brazil and New Zealand by Arnaldo Carvalho de Melo and the author using the public Internet which consisted of a route using 18 hops. The tests were done with netcat and ttcp which had been modified to use DCCP. Tests were carried out by others with success also which demonstrates that ISPs are allowing DCCP to traverse their networks. That is probably due to it being built on top of IP, and most IP traffic being allowed.

Initially the tests failed with checksum failure on the header. This was proven to be due to Network Address and Port Translation (NAPT) as the DCCP checksum covers the source and destination IP addresses amongst other fields, and NAPT changes the source IP and port address. With checksum tests temporarily removed the tests proceeded successfully. It is extremely encouraging to see that DCCP was able to traverse the public Internet successfully. After these tests were completed the Linux firewall code was modified to rewrite the DCCP checksum during NAPT, in the same manner as is done for TCP and UDP. As future Linux based firewalls use newer code this will improve the suitability of DCCP for use on the Internet. During the course of this research I was also contacted by people from a range of firms such as Sony, multimedia startups and military subcontractors which indicates that there is some interest in DCCP being implemented. DCCP can also be encapsulated over UDP and this is an RFC [93] at Proposed Standard status. TCP will still be used by applications where reliability is needed, such as control information and this is seen also for applications that use UDP.

An interesting effect was observed when testing with the SBPN algorithm that sorts the transmit in priority order and discards all expired packets. The first packet was transmitted with an RTT of 0 milliseconds and after that

```
ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
```

Figure 4.1: sendmsg system call function prototype in Linux

the RTT was 500 milliseconds [40] until the RTT was observed from return packets. The actual RTT on the public Internet test was 130 milliseconds. This meant that packets were discarded by the SBPN algorithm as unable to reach the destination in time until the real RTT was obtained. DCCP was modified to fix this by following a suggestion from RFC3448 erratum to get the RTT from the initial handshake establishing the session.

The initial implementation of DCCP had no limit to the size of the transmit queue. The result of this was that if the transmit rate of the application exceeded the transmit rate of the network, packets would get more and more delayed. This is clearly not desirable, and could also lead to out of memory crashes. Transmission queue size limits were implemented as a limit on the number of packets waiting to be transmitted, and an error is returned to the application if the queue is full and a further packet is requested to be transmitted.

4.3 Implementation of SBPN in Linux DCCP

SBPN was implemented by making a number of changes to the standard Linux DCCP implementation. These patches [74] implemented SBPN in the Linux kernel and created an API to allow applications to pass information to SBPN.

The SBPN API uses the sendmsg system call shown in Figure 4.1 to allow the application to send data to the SBPN implementation in the Linux kernel. The sendmsg function in Linux allows control data to be passed to the kernel alongside the data itself in the msghdr structure as shown in Figure 4.2.

For SBPN we provide the dccp_prio structure shown in Figure 4.3. This is passed into the msg_control field as the control data which is set aside for applications to pass data to the operating system. In this structure the expiry

```
struct msghdr {
             void         *msg_name;
             socklen_t     msg_namelen;
             struct iovec *msg_iov;
             size_t        msg_iovlen;
             void         *msg_control;
             size_t        msg_controllen;
             int           msg_flags;
              };
```

Figure 4.2: msghdr structure for sendmsg system call

field contains the time after which the data will be of no use to the receiver, the priority field which specifies whether the data contains audio or video, and the method field is used to select which SBPN algorithm will be used in testing. These fields are used for all the testing of algorithm variants except when unmodified DCCP is being tested.

The data to be transmitted and the control data are both received by the dccp_sendmsg function from the sendmsg function. The dccp_sendmsg function uses the priority information to store the packet in the correct location in the transmit queue. The transmit queue is a single queue but was changed from a FIFO queue to a queue sorted first by priority, and secondly by expiry time. This was implemented as one queue for reasons of code simplicity, but for future research it could also be implemented as a set of queues.

The dccp_write_xmit function carries out transmission of packets for DCCP in Linux and this is where the transmit part of the SBPN algorithm is implemented as described in Section 3.3. This function starts transmission from the beginning of the priority queue. The function checks the expiry time just before a packet is transmitted and decides whether or not to discard or transmit it as per the SBPN algorithm.

```
struct dccp_prio {
                struct timeval expiry;
                u16 priority;
                u16 method;
                   };
```

Figure 4.3: dccp_prio structure passed in msg_control field

4.4 Video conference model

To gain an understanding of real time media applications network usage, current application audio and video traffic was captured and analysed. Once the data was analysed a model combining the findings was created. This model allows us to perform simulations based on existing programs, and thus see how SBPN would potentially improve user experience.

4.4.1 Data captured

Data was captured from three applications — Skype, MSN Messenger and Ekiga. Skype and MSN Messenger were running on Microsoft Windows and Ekiga was running on Linux. Skype and MSN Messenger were chosen as these were in common use in mid 2006 when the analysis was carried out. Ekiga was chosen as a commonly used program on Linux in mid 2006 and to allow further analysis to be carried out if needed by using Linux tools, and to allow access to any documentation or source code if needed as Ekiga is an open source program.

The traces were captured on a 100 Megabit per second network link with both sender and receiver connected to the same network switch to ensure that there was enough bandwidth for any codec and virtually no latency. The reason for this was so that the data would be captured on an uncongested link as we wanted to see the underlying traffic model, rather than how the programs responded to congestion. The traces were run for twenty seconds and consisted of ten seconds of talking and motion, followed by five seconds of silence and no motion, followed by five seconds of talking and motion. The

video component was a person sitting in front of a webcam, and the motion was arms being waved in the air and the periods of no motion were the person sitting as still as possible.

The packets in the traces were then analysed to see their protocol, timing between packets, packet size, and to try and determine whether each packet contained audio or video. When analysing the traces it could be seen that UDP was used by all the applications to transmit the audio and video packets. A minimal amount of TCP was used in the connection setup and, presumably, for control purposes due to a requirement for reliability. The TCP traffic was excluded from our analysis and model generation as the TCP packets constituted less than 0.1% of the total network packets.

4.4.2 Ekiga results

The Ekiga voice packets had a very regular pattern when not carrying out silence suppression. All packets are 214 bytes with a pattern of six packets spread over 120 milliseconds with the sequence of intervals between packets being 39, 9, 15, 24, 24, 9 milliseconds with little variation to this pattern. Ekiga's audio is transmitted using the H.323 standard according to their documentation. The audio was transmitted on UDP port 5004. In periods of silence suppression no audio packets were sent at all. The packet spacing being not at exactly the same intervals may be their method of congestion control and designed to avoid synchronisation issues if many streams were all sending at the same regular time intervals.

The video component sends a number of packets which appear to constitute a video frame together every 100 milliseconds on UDP port 5006. The maximum size of the frame is 6677 bytes and the minimum size is 155 bytes. The frame size appears to be totally differential (transmitting the difference between frames) as there is no discernible pattern of large frames at repeated intervals and the trend for frame sizes matches the amount of motion. For Ekiga with high motion the average is 4451 bytes per video frame with a stan-

dard deviation of 1014 bytes, while for low motion the average is 1072 bytes with a standard deviation of 482 bytes. The maximum size of a packet within a frame is 1078 bytes. For packets that are not at the end of a frame the average is 1026 bytes with a standard deviation of 44 bytes. The last packet in a frame is of variable size to complete the transmission of a frame. In 20 seconds of video there were a total of 799 packets for 200 frames.

Tests were also carried out on how Ekiga responded to congestion. Initial tests showed that traffic for voice is not prioritised over that for data. In testing Ekiga was restricted to a bandwidth allocation that was less than the sum of the bandwidth required for video and audio, but higher than that required for either component by itself. The result of this was that the audio came through at a different pitch and less intelligible indicating that not all audio packets were being received. This shows the potential for real time media applications such as Ekiga to improve the quality of user experience through methods such as SBPN.

4.4.3 MSN Messenger results

For MSN Messenger audio the average time between packets is 20 milliseconds which equates to 50 packets per second. The packets arrive in sequences of five generally with a pattern of four packets at larger intervals (23.6 milliseconds) followed by a packet with a shorter interval (average 3.5 milliseconds). The percentage of packets with less than 5 milliseconds between packets is 18.5% and the percentage of packets with larger than 9 milliseconds between packets is 81.5%. There were no packets with a gap between packets in the range of 5 milliseconds and 9 milliseconds. These figures are all for audio when no silence suppression was occurring.

MSN Messenger did use silence suppression for audio during the quiet periods. When silence suppression occurred 78 byte packets are sent and time between packets is longer than other audio packets. The time between packets can extend to multiple seconds. At other times the audio packets range be-

Table 4.1: Skype traffic characteristics

Packet size Range (bytes)	Packet size Average (bytes)	Packet size Std deviation (bytes)	Time interval Average (msecs)	Time interval Std deviation (msecs)
100-339	235	57	30.1	2.5
459-799	644	80	27.9	12.1
890-1294	1061	74	16.7	7.6

tween 124 and 160 bytes with a large number of packets with 125 bytes. The average packet size is 123 bytes with silence suppression or 133 bytes without silence suppression. There is a noticeable trend after periods of 78 byte packets that the packets become larger and are closer to 160 bytes in size.

For video MSN Messenger sent frames on average every 40 milliseconds with a frame size during high motion of 1400 bytes with a standard deviation of 275 bytes, and for periods of low motion of 650 bytes with a standard deviation of 135 bytes. A frame was either sent as one packet or broken into smaller packets with the first packet being either 896 or 1066 bytes. There was a gap in transmission time between packets in a frame indicating that MSN Messenger was employing some form of pacing control or congestion control over top of UDP.

4.4.4 Skype results

With Skype it was harder to carry out analysis and determine which were audio packets, and which were video packets as all packets were sent over the same UDP port. The packet distribution shows what appears to be patterns for different sizes of packets and this is shown in Table 4.1 with very few packets outside these ranges.

Assuming that the smaller packets were voice then Skype voice packets ranged from 108 bytes to 335 bytes with an average of 252 bytes and a standard deviation of 42 bytes, after removing one outlying packet from the data. These packets were sent every 30 milliseconds which would also validate the suggestion that these are audio packets as audio packets need to be sent at

a higher frequency than video packets. Packets were still sent during periods of silence, so it would appear that Skype did not carry out silence suppression. It is interesting to break packets down into periods of active interaction and silence and these show that during active periods the average packet size is 258 bytes with standard deviation 45 bytes, and during silence the average packet size is 234 bytes with standard deviation 18 bytes. These smaller packets sizes during silent periods may however not be statistically significant. Further investigation was not carried out.

Skype has some form of timing control for video as packets are not sent immediately after each other, but always had a gap between each packet even in times of high motion video when a video frame would be larger than a single packet. It is a possibility that the packets in the largest size range were mostly for frames, and that medium size packets were used to complete frames, and that the variance in small packet sizes is when video data is combined with audio data. This would also match the fact that all data is transmitted on the same port.

4.4.5 Traffic model

A model of the traffic was then built based mostly on how Ekiga transmits packets, although the applications were reasonably similar in their characteristics. Fixed rate audio was used with a payload size of 214 bytes every 20 milliseconds and silence suppression was not used. Video frames in the model were created with an average and a standard deviation for their type that matched the traffic Ekiga generated as detailed in Section 4.4.2. During periods of high activity the average packet size is 4451 with a standard deviation of 1014. During periods of low activity the average packet size is 1072 with a standard deviation of 482.

With this model audio equates to 86 Kbps and video equates to 289 Kbps which gives a total of 374 Kbps. These rates are inclusive of all packet overheads.

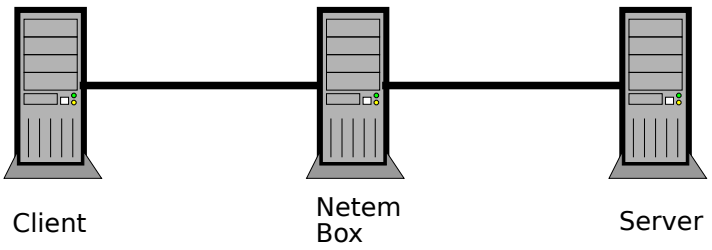


Figure 4.4: Test network for SBPN testing - Netem installed on middle box

It was decided not to incorporate silence suppression into the model. The reason for this is that SBPN is trying to demonstrate improvements in the first instance for audio performance.

4.5 Building of test framework

4.5.1 Netem

To generate loss on a link a tool was needed and Netem was chosen.

Netem is an open source traffic shaping tool developed by Stephen Hemminger and included in the Linux kernel from version 2.6.7 onwards. Netem can be used to drop packets, inject delay, duplicate packets and reorder packets. The distribution on the loss, duplication and delay can be uniform or user defined. Netem draws ideas from Dummynet which is part of FreeBSD and some code from NIST Net [82].

Netem only works on outbound queues, and not inbound queues, so to achieve symmetric loss on a link it is necessary to enable it on a PC running as a router with at least two Ethernet cards. The setup used for DCCP testing in this thesis with Netem is shown in Figure 4.4. This setup is also useful to minimise interaction with other Linux network code, as in early testing it was found to produce unexpected results if run on the same box as the client and server software.

The use of Netem was validated by using it to vary loss and delay to see if the throughput matched the rate predicted in DCCP CCID3 with the TFRC equation. This actually highlighted bugs in the CCID3 code that were fixed

before further testing was carried out.

4.5.2 Test applications

A suite of test applications were created to carry out testing of SBPN with the model created.

The tests were run with all unnecessary services such as X [13] and NFS [103] shut down, which reduced the baseline CPU usage to nearly zero and all tests were run from a text console. This enabled the testing programs to have full access to the CPU without interference from other programs.

The kernel on the test machines were also built with a 'HZ' setting of 1000. This setting means that the CPU scheduler is run 1000 times per second - i.e. at 1 millisecond intervals. The 1 millisecond resolution means that tasks are scheduled with reduced variability compared to that with 10 millisecond scheduling which was the Linux kernel default at the time of the tests. With tests being carried out with expiry times of between 15 and 75 milliseconds this reduces the variability due to scheduling of timeslices for the userspace components of the test suite.

From the model generated in the previous section a Comma Separated Values (CSV) formatted text file was generated which contained a list of packets to be sent, their content, size and timing. An application was written to then replay this stream between two PCs with a Linux PC in the middle using the Netem software to add delay, and to rate limit the traffic or create loss. All of the packets in an audio frame or video frame were sent immediately, one after the other by the test application so the actual time to send was determined solely by the DCCP CCID3 congestion control in the Linux operating system. This differs from applications which use UDP, as some of these determined the send time themselves as UDP does not have congestion control at the transport level as detailed in Section 4.4 and they added their own packet timing as a form of congestion control. If audio and video frames were both due to be transmitted at the same time the application randomly chose which

to send first to avoid bias as typically audio and video are handled by separate threads in a program and either could take priority. The result was a data stream that in effect shifted any timing decisions from the application to the network transport layer, so that it could be seen if improvements could be made there.

There was a separate application written to track the difference in time setting between the two PCs so that arrival time could be tracked at the receiver relative to the sender. The receiver recorded how long each packet took to arrive and whether it arrived at all. On the receiver the identity and timing of the packets received were written to a CSV format file so that further analysis could be carried out. To check whether packets had arrived on time initially it was thought that using Network Time Protocol (NTP) on the sender and receiver would enable both machines to synchronise their time. When analysing initial test data it was established that this assumption was incorrect as NTP did not synchronise time to lower than 1 millisecond and was often over 50 milliseconds. The reason for the inaccuracy that occurred is that NTP requires accurate clocks and uncongested networks to keep time to within 1 millisecond. The clocks used were not accurate on the PCs and would drift as much as 2500 milliseconds in a series of runs and the network links had delay as part of the testing on them so NTP would not have synchronised them even if it had the capability to do it continuously at high frequency and accuracy. A program was written to establish the time difference between two machines that are connected via a network. The program did this by sending a packet from one machine to the other with the time in it, and the other machine sending back it's time. The RTT could then be calculated, and the time compared between the two nodes with half the RTT deducted (as this is the one way delay). This program was then used when calculating results to ensure that the time taken to send and receive packets could be calculated to within 1 millisecond.

The length of the audio and video test in Section 4.4.1 was 20 seconds. Ini-

tial testing with the 20 second runs however showed a high rate of variability due to the initial connection setup and the random nature of loss and congestion. At low loss rates in particular this altered the timing of sending rate alterations in proportion to the length of a run. The test was then increased to 60 second runs with the data being repeated three times in each run. The first 3 seconds of data was discarded and the remaining data had significantly reduced variation due to the connection speed stabilising over the longer run. In all of the results in this thesis data was generated from 30 runs of 60 seconds to ensure that the results were statistically significant. An average was calculated with 95% confidence intervals from a T-test as recommended in Law [70] and the confidence intervals are shown on all the graphs in this thesis.

Huffaker [52] has measurements on RTTs and our times of 30, 80 and 150 milliseconds correspond with their ranges of: on the same coast of the USA; between coasts of the USA; and intercontinental from the USA. These measurements were used to run tests with symmetric one way delays of 15, 40, 75 milliseconds producing minimum RTTs of 30, 80, 150 milliseconds. For the purpose of simplicity symmetric links were used. It is recognised that not all links are symmetric [89]. The algorithm could easily be adapted if the one way delay could be measured for a link.

4.6 Measuring audio quality

To determine whether SBPN is making a difference to audio quality it is important to quantitatively measure the audio quality. In this thesis two measures are used — on time arrival (Section 4.6.1) and MOS (Section 4.6.2). These are described below. On time arrival measures whether a packet arrives within a certain time period, and MOS measures the quality of audio as would be perceived by a human listener.

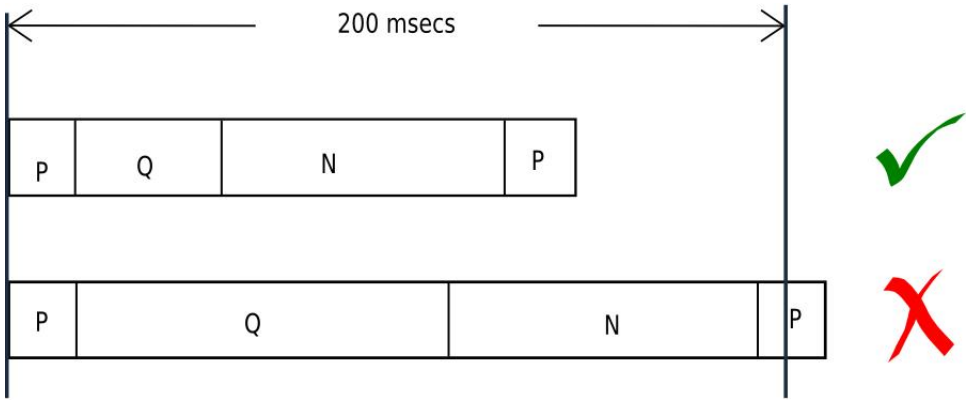


Figure 4.5: Diagram illustrating on-time arrival

4.6.1 On time arrival

The ITU recommend that delivery time for audio be less than 150 milliseconds, and that any delivery time between 150 milliseconds and 400 milliseconds has lower quality [58] than with the recommended delivery time. The reason for this recommendation is that the longer the delay, the harder it is to have a conversation as there are delays in answering the other person and people can both talk at the same time and take longer to realise that this is occurring. With SBPN, packets get marked with an expiry time which is 200 milliseconds from the time that they are created. The expiry time includes both time in the queue and time in transit. The choice of 200 milliseconds is arbitrary and it is envisaged that an application could allow the end user to adjust this setting to meet their quality requirements.

In this thesis when a packet arrives within the expiry time of 200 milliseconds the packet is said to have arrived on time for the experiments. The desired result is that more audio packets arrive on time. This is measured through the percentage of packets that arrive on time.

Figure 4.5 shows the timing graphically where P is processing time (both server and client) and includes tasks such as frame encoding and decoding, Q is the time spent in the queue before transmission time and N is the time taken to traverse the network. The top box shows a packet arriving on time as all steps occur in less than 200 milliseconds, and the bottom box shows a packet

Table 4.2: MOS Score and Quality description

MOS Score	Quality
4.34-4.50	Best
4.03-4.34	High
3.60-4.03	Medium
3.10-3.60	Low
2.58-3.10	Poor

that does not arrive on time as the total time of all steps is greater than 200 milliseconds.

4.6.2 Mean Opinion Score

ITU recommendation G.107 [59] introduces the E-model - "A computational model for use in transmission planning" which has a quality measure R which is transformed into Mean Opinion Score (MOS). MOS gives a measure that correlates to how people rate the audio quality. MOS is used for measuring the quality of telephone conversations. The MOS score and the corresponding quality rating are shown in Table 4.2, and the equations to calculate these as per the ITU recommendations are detailed below:

The value of MOS is calculated in Equation 4.1 that takes the value of R calculated in Equation 4.3 as the input. If R is less than 0 then it is clamped to a minimum of 1, and if it is greater than 100 then it is clamped to a maximum of 4.5.

$$\begin{aligned} &\text{if } R < 0,\ MOS = 1 \\ &\text{if } R > 100,\ MOS = 4.5 \\ &\text{else } MOS = 1 + 0.035R + 7 \times 10^{-6} R \times (R - 60) \times (100 - R) \end{aligned} \tag{4.1}$$

The starting point for R is calculated in Equation 4.2 where I_s is a measure related to signal-to-noise, I_d is a measure related to ear-to-mouth delay, I_{ef} is an equipment impairment factor measure and A is an Expectation Factor. The Expectation Factor is a measure to adjust for people changing their expec-

tations depending on what they know about the call e.g. people may expect international or cellphone calls to be of lower quality.

$$R = 100 - I_s - I_d - I_{ef} - A \tag{4.2}$$

Cole [26] takes Equation 4.2 and considers it for the case of VoIP and the resulting equation is shown in Equation 4.3 where I_e is the impact of loss and I_d is impact of delay. The calculation for I_e is shown in Equation 4.4. $\lambda_1, \lambda_2, \lambda_3$ are dependent on the codec used and e is the loss rate. In the case of the G.711 [57] codec the values from Balan [31] are $\lambda_1 = 0, \lambda_2 = 30, \lambda_3 = 15$. I_d is calculated in Equation 4.5 where d is the delay in milliseconds. The more delay that occurs the higher the impact, and this increases at a faster rate after 177.3 milliseconds.

$$R = 94.2 - I_e - I_d \tag{4.3}$$

$$I_e = \lambda_1 + \lambda_2 ln(1 + \lambda_3 e) \tag{4.4}$$

$$\begin{gathered} \text{if } d < 177.3,\ I_d = 0.024d \\ \text{else } I_d = 0.024d + 0.11(d - 177.3) \end{gathered} \tag{4.5}$$

G.711 was chosen as the codec because the bandwidth for the model in Section 4.4 most closely matched the bandwidth of the model developed in Balan [31]. The parameters that vary between codecs are λ_1, λ_2 and λ_3. As discussed above Equation 4.3 shows that R score is dependent on delay and loss. In Equation 4.5 there are no codec dependent values, and Equation 4.4 is comprised of constants for a codec and the loss rate. As such the choice of G.711 as the codec will not alter whether the SBPN algorithm will make an improvement or not, but will only alter the size of the improvement that is

made when compared to another codec.

4.7 Summary

In this chapter the tools needed to create and test SBPN, and how these were constructed were outlined. These are important steps to take the SBPN from an idea to an actual platform that can be tested and measured, and to see whether SBPN actually makes an improvement.

The steps needed to get DCCP on Linux into a state able to be used were described in Section 4.2 as prior to this research there was not an effective DCCP implementation available on Linux.

Section 4.3 describes the API for SBPN and how this was implemented on the Linux kernel. For SBPN to be used in actual programs there needs to be a mechanism for programs to use it and the interface that was created is outlined here.

Section 4.4 described the development of a video conferencing model, that was made by reverse engineering three different application's traffic flow. These traffic flows were then analysed and used to create a model. This allowed testing to be carried out on traffic based on actual user applications.

Section 4.5 describes the tools that were created and used for testing SBPN. Netem is used to vary network conditions such as speed and loss. Tools were written to enable measurements to be made, and the steps needed to create an effective testing environment were detailed. This was important to allow tests to be controlled and repeatable.

It is important to be able to measure any improvements made objectively so that it can be determined if SBPN makes a difference to real time media applications. Section 4.6 proposes how to evaluate the quality improvements being made by SBPN by measuring on time arrival and Mean Opinion Score (MOS). On time arrival describes how many packets arrive within a certain time period (200 milliseconds), as delayed audio causes worse experience for

the person listening and MOS quantifies how a person would rate the quality of audio on a scale.

The work to implement SBPN on top of DCCP for Linux, the video conferencing model, and quality measures that will be used to determine whether SBPN has made an improvement have been described. The following chapter begins to outline test results.

Chapter 5

Results from testing SBPN

5.1 Introduction

In the previous chapters the need to improve real time media application audio performance was discussed, solutions were proposed and their implementation described along with ways to measure audio quality.

In this chapter results are presented from testing SBPN. Results are shown graphically with on-time arrival and the quality measured by Mean Opinion Score (MOS) across a range of scenarios. Explanations are given for the results and different tests are carried out to see if improvements can be made.

In Section 5.2 testing is carried out with loss on the network. SBPN1 is tested in Section 5.2.1 firstly with an unbounded length queue and then with a fixed length queue. SBPN2 is tested in Section 5.2.2 to see if it can improve on SBPN1 and in Section 5.2.3 analysis is carried out on packets transmitted to understand the behaviour of SBPN under lossy conditions.

Testing with congestion is detailed in Section 5.3. Section 5.3.1 gives results for testing SBPN2 with congestion and further testing is described in Section 5.3.2 for varying queue lengths and Section 5.3.3 for varying RTTs.

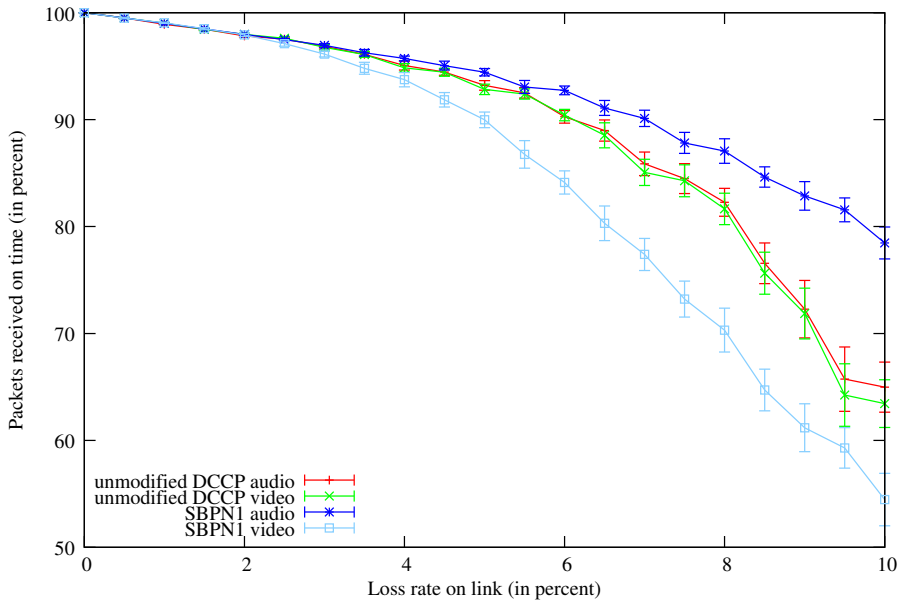


Figure 5.1: Testing with loss - 80 ms RTT, unbounded queue

5.2 Testing with loss

The first testing that was done with SBPN was with loss created on the network. This section outlines the testing and the results. It was decided to test with loss, as often loss is an indicator of congestion on a network, although it can also indicate factors such as packet corruption.

5.2.1 SBPN1

5.2.1.1 Testing SBPN1 with an unbounded queue

In the first tests carried out unmodified DCCP packet transmission is compared to DCCP with the SBPN1 algorithm implemented. An unlimited length transmission queue is used (for both unmodified DCCP and SBPN1) as this was the only option in the Linux implementation at the time of initial testing. Unmodified DCCP and SBPN1 were tested on a network using Netem to create delay and to create loss by discarding packets as described in Section 4.5. Netem was configured to drop random packets without any correlation between packets dropped.

The tests were carried out with an 80 millisecond RTT on the network

(40 milliseconds of delay was created by Netem) and the results are shown in Figure 5.1. Tests were run for 60 seconds and repeated 30 times as detailed in Section 4.5.2. With the SBPN1 algorithm, when compared to normal transmission, a higher proportion of audio packets arrive on time with a corresponding reduction of video packets arriving on time. The vertical line at each data point shows the range for a 95% confidence interval from a T-test. The audio data for this shows that at 10% loss on the link, using unmodified DCCP results in 198 millisecond average delay with 10.0% packet loss and using SBPN1 results in 85.4 millisecond average delay with 21.5% of packets lost or discarded for SBPN1. The much higher rate of packet loss results in a drop of quality that outweighs the reduction in average delay. For video packets there were 45.5% of packets lost with SBPN1 and 99.2 millisecond average delay.

The rate of audio packets arriving on time rose from 65.0% with unmodified DCCP to 78.5% with SBPN1. Although fewer packets were transmitted with SBPN1 all of the packets that arrived for audio were on time. There is also a corresponding drop in the number of video packets arriving on time as this falls from 63.4% with unmodified DCCP, to 54.5% with SBPN1.

The rate of packets not received ontime with SBPN1 is not equal to the loss rate as the loss rate increases. This is because audio packets are not transmitted as they will expire in transit. The proportion of video packets transmitted is lower than the proportion of audio packets transmitted, and becomes lower as the loss rate increases. Some video packets are still transmitted though, even at higher loss rates, as the last audio packet in the queue may be discarded and when the next packet transmission is allowed no further audio packets have been queued.

Figure 5.2 shows the results from a quality point of view. MOS is calculated using the average figures from the testing runs. MOS is calculated as described in Section 4.6.2 and is based heavily on the impact of loss and delay on packets. In this figure the unmodified DCCP shows higher quality results than SBPN1

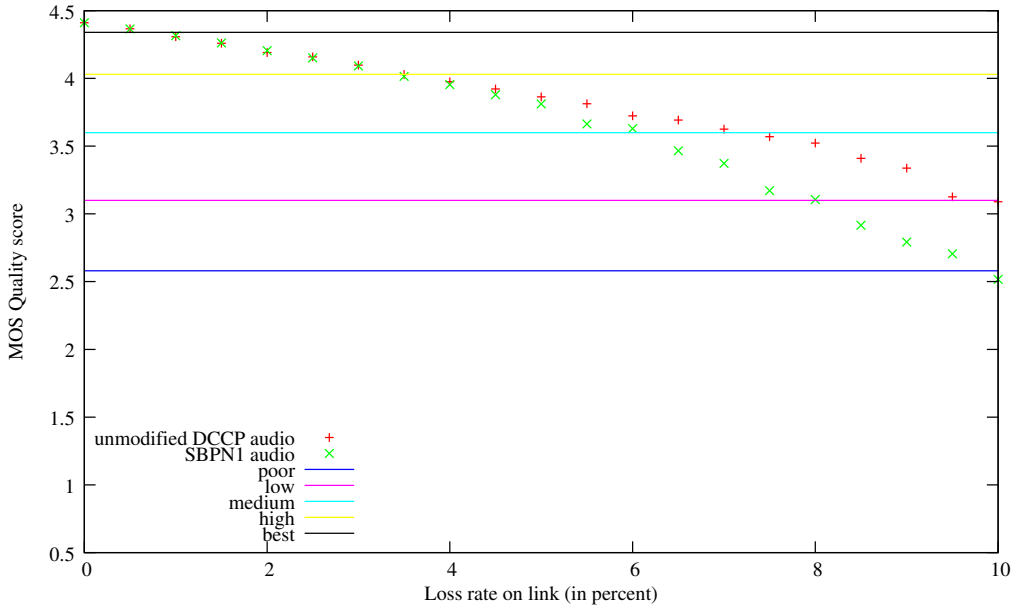


Figure 5.2: Testing with loss - 80 ms RTT, unbounded queue

which is not the desired result. Audio quality is expected to be improved with SBPN.

Table 5.1 show the underlying data used to calculate the quality metrics in Figures 5.2. Rate is the constrained link rate in Kbps, loss is the percentage of packets that did not arrive at the receiver. I_e is the impact of loss as defined in Equation 4.4, I_d is the impact of delay as defined in Equation 4.5, and MOS is the Mean Opinion Score as defined in Equation 4.1.

Looking at the data in the table it can be seen that the lower quality is due mostly to the variance in the impact of loss (I_e), which is derived from the loss rate. For example at 10% loss on the link the loss rate increases with SBPN1 from 9.9% to 21.5% over unmodified DCCP. At 10% loss the average delay drops from 198 milliseconds to 43.3 milliseconds but the decrease in I_d (impact of delay) is more than offset by the increase in I_e (impact of loss).

5.2.1.2 Testing SBPN1 with a fixed length queue

When this test was carried out the Linux implementation of DCCP had an unlimited transmit queue, and it was wondered if this was part of the reason for SBPN1 being worse than unmodified DCCP. Having an unlimited transmit

Table 5.1: Loss and delay with unbounded queue length, 80 ms RTT

	Unmodified DCCP					SBPN1				
Link Loss	Packet Loss	Delay	I_e	I_d	MOS	Packet Loss	Delay	I_e	I_d	MOS
0.0%	0.0%	40	0.0	0.9	4.4	0.0%	40	0.0	1.0	4.4
0.5%	0.5%	40	2.1	1.0	4.4	0.5%	40	2.1	1.0	4.4
1.0%	1.1%	40	4.5	1.0	4.3	1.0%	40	4.2	1.0	4.3
1.5%	1.5%	41	6.2	1.0	4.3	1.5%	40	6.1	1.0	4.3
2.0%	2.2%	43	8.4	1.0	4.2	2.0%	42	8.0	1.0	4.2
2.5%	2.4%	45	9.3	1.1	4.2	2.5%	42	9.6	1.0	4.2
3.0%	3.0%	47	11.0	1.1	4.1	3.1%	44	11.3	1.1	4.1
3.5%	3.6%	49	12.9	1.2	4.0	3.8%	45	13.3	1.1	4.0
4.0%	4.0%	54	14.2	1.3	4.0	4.3%	47	14.9	1.1	4.0
4.5%	4.5%	56	15.5	1.4	3.9	4.9%	49	16.6	1.2	3.9
5.0%	5.0%	61	16.8	1.5	3.9	5.6%	51	18.2	1.2	3.8
5.5%	5.4%	63	17.9	1.5	3.8	6.9%	56	21.4	1.3	3.7
6.0%	6.2%	71	19.7	1.7	3.7	7.3%	58	22.1	1.4	3.6
6.5%	6.4%	78	20.2	1.9	3.7	8.9%	62	25.4	1.5	3.5
7.0%	7.0%	89	21.5	2.1	3.6	9.9%	65	27.3	1.6	3.4
7.5%	7.4%	98	22.4	2.4	3.6	12.2%	69	31.2	1.6	3.2
8.0%	7.8%	107	23.2	2.6	3.5	12.9%	72	32.3	1.7	3.1
8.5%	8.6%	138	24.8	3.3	3.4	15.4%	77	35.9	1.9	2.9
9.0%	8.9%	167	25.5	4.0	3.3	17.1%	80	38.2	1.9	2.8
9.5%	9.5%	198	26.6	7.1	3.1	18.4%	81	39.8	1.9	2.7
10.0%	9.9%	198	27.4	7.0	3.1	21.5%	85	43.3	2.0	2.5

queue is not a realistic scenario as other protocols have maximum buffer or queue sizes. This reflected the status of implementation of DCCP in the Linux kernel at the time. An unbounded queue can potentially grow to a point where it causes issues with the operating system due to memory consumption. An unbounded queue may also have a growing delay to transmission. If the transmit rate is lower than the queueing rate, the queue will continue to grow in length and packets that are transmitted will progressively have a longer delay between queuing and transmission.

A limited transmission queue was implemented in the Linux kernel and the test for SBPN1 was carried out again, but with a maximum queue length of five packets. A length of five packets was chosen as this was observed to be the average maximum UDP queue length when existing applications were analysed in Section 4.4.

When run under these conditions the SBPN1 algorithm performed worse than unmodified DCCP as shown in Figure 5.3 and Figure 5.4. Audio and video packet on time arrivals both decreased for SBPN1 compared to unmodi-

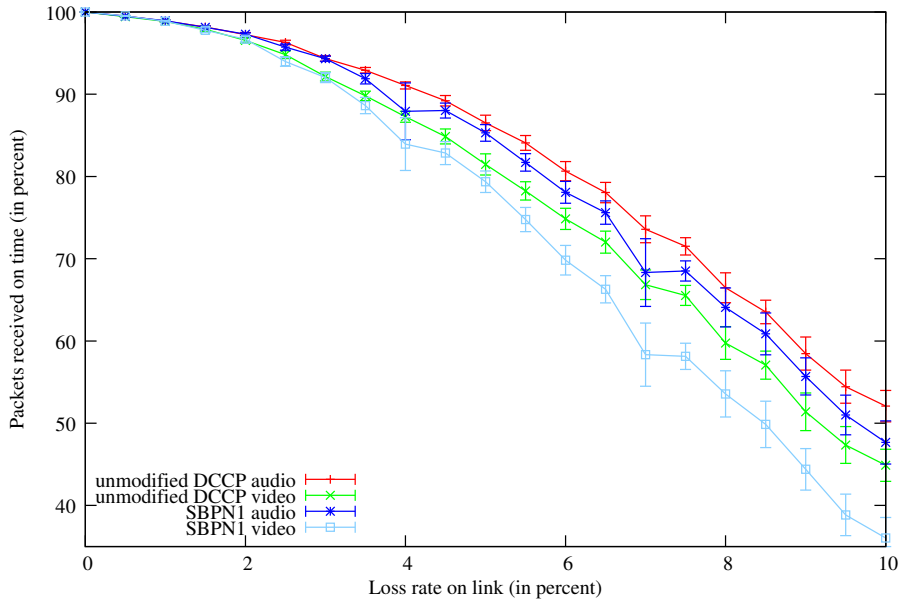


Figure 5.3: Testing with loss - 80 ms RTT, queue length 5

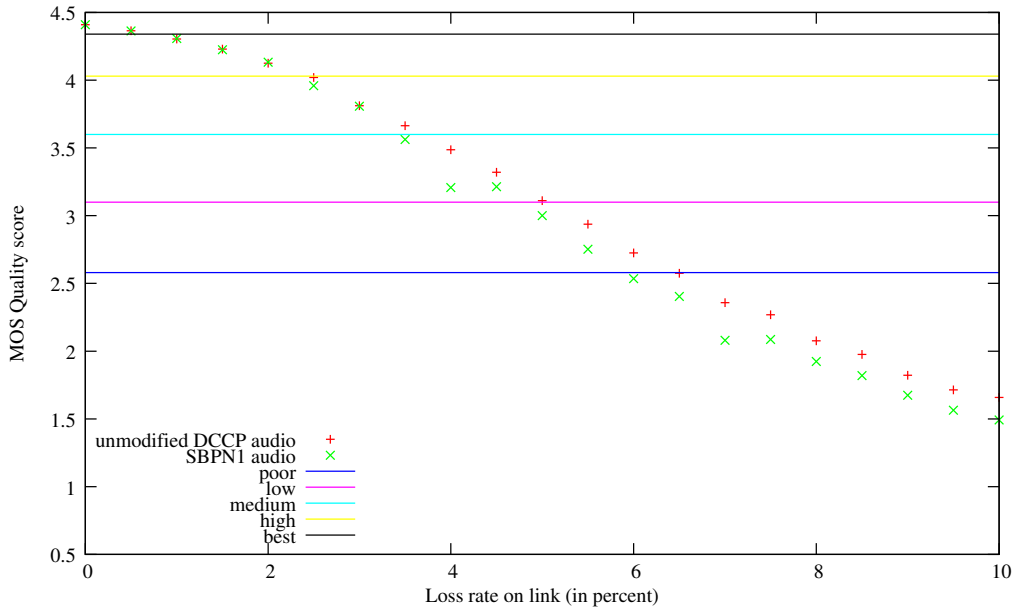


Figure 5.4: Testing with loss - 80 ms RTT, queue length 5

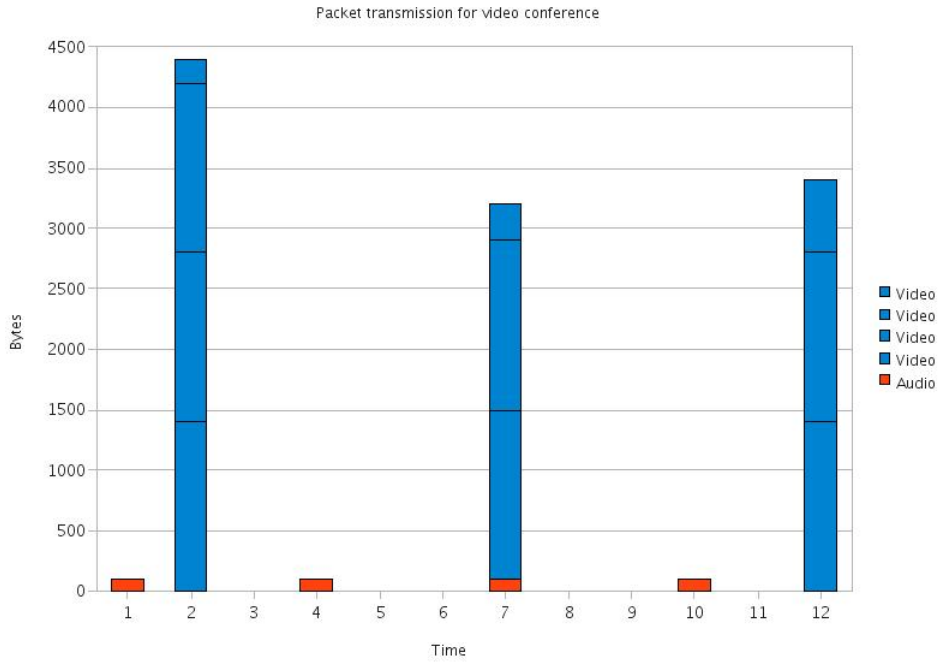


Figure 5.5: Timing of audio and video packet transmission

fied DCCP and quality measures were also worse. The data for this shows that at 10% loss on the link, using unmodified DCCP resulted in 91 millisecond average delay with 44.0% packet loss and using SBPN1 results in 72 millisecond average delay with 52.3% packet loss for SBPN1. The higher rate of packet loss again results in a drop of quality that outweighs the reduction in average delay. It can however be seen that the reduction in quality is less than with unlimited transmit queue as shown in Figure 5.2.

In Figure 5.3 it can be seen that more audio packets arrive on time than video packets with unmodified DCCP, which differs from Figure 5.1 where they are approximately the same. The reason for this is that all packets for a single video frame get queued for transmission at the same time which means there is a higher probability that a video packet is discarded when the buffer is full. This is based on the application discarding the data if the queue is full. The results might differ if a "blocking" architecture was used for packet queuing where the operating system waits for a queue entry to become free before returning control to the application. Figure 5.5 shows an example of this packet transmission timing for video conferencing graphically.

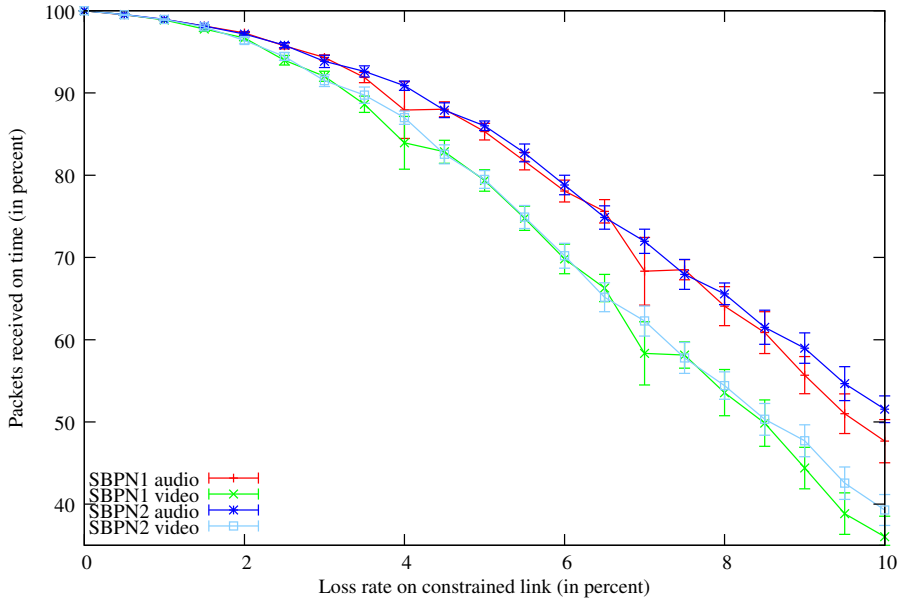


Figure 5.6: Testing with loss - 80 ms RTT, queue length 5

The results for SBPN1 were worse than was expected when the algorithm was designed. With SBPN1 a packet is always discarded if the packet will reach the receiver after its deadline – the packet has expired. This problem was discussed further in Section 3.5 and is due to the fact that TFRC reduces the sending rate allowed, based on the previous sending rate. So instead of a virtuous circle being created based on SBPN1 discarding useless data, a vicious circle is created where allowed sending rate is decreased because SBPN1 reduces its transmission rate.

5.2.2 Testing SBPN2

The SBPN2 algorithm was introduced in Section 3.5 to improve on the issues found through practical experiments in the previous section. With SBPN2 expired packets are only discarded if the transmit buffer has other packets in it. As discussed in Section 3.5, TFRC reduces the allowable transmit rate if packets are not being transmitted. SBPN2 aims to reduce this effect by discarding fewer packets than SBPN1.

Testing was carried out with the SBPN2 to compare it to SBPN1. The results with queue length of 5 packets and 80 milliseconds RTT can be seen in

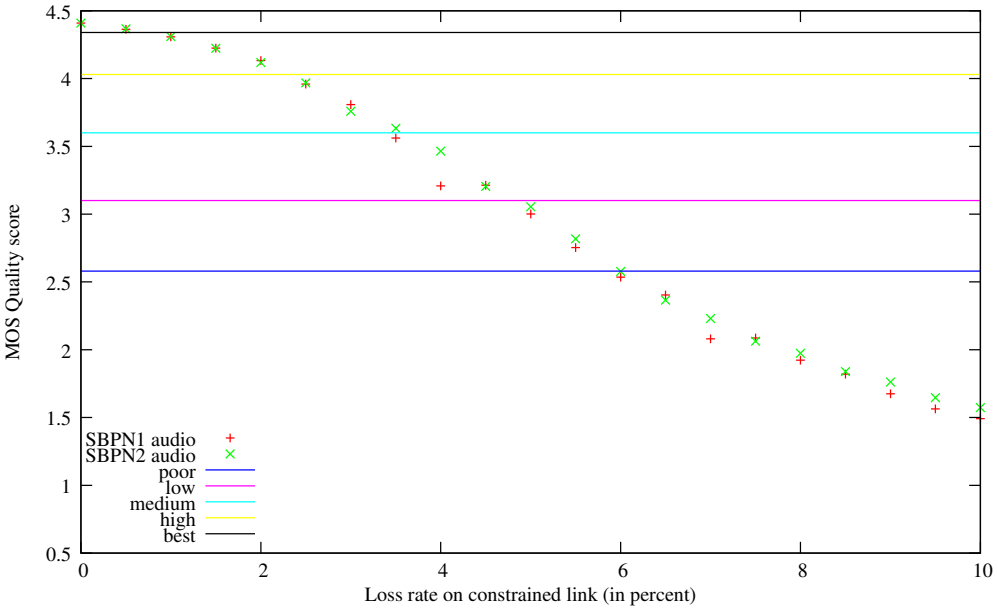


Figure 5.7: Testing with loss - 80 ms RTT, queue length 5

Figure 5.6 and Figure 5.7 which show packets received on time and the MOS quality score respectively. These graphs show that packets received on time and quality both improved slightly.

The underlying data used to calculate the MOS quality score shows that, for example, at 10% loss on the link, using SBPN1 results in 72 millisecond average delay with 52.3% packet loss and using SBPN2 results in 72 millisecond average delay with 48.4% packet loss. This shows that with SBPN2 the rate of packets lost did decrease without any change in the average delay and this resulted in the increased quality.

In Figure 5.8 and Figure 5.9 unmodified DCCP is compared to SBPN2. This improved results compared to the SBPN1 algorithm but did not achieve results that were better than unmodified DCCP. This shows that sending more packets did improve the on-time arrival and quality so further analysis was carried out.

5.2.3 Analysing packets transmitted

Figure 5.10 shows the difference in percentage between the data received, and the data received on time. In the graph SBPN1 has almost no difference be-

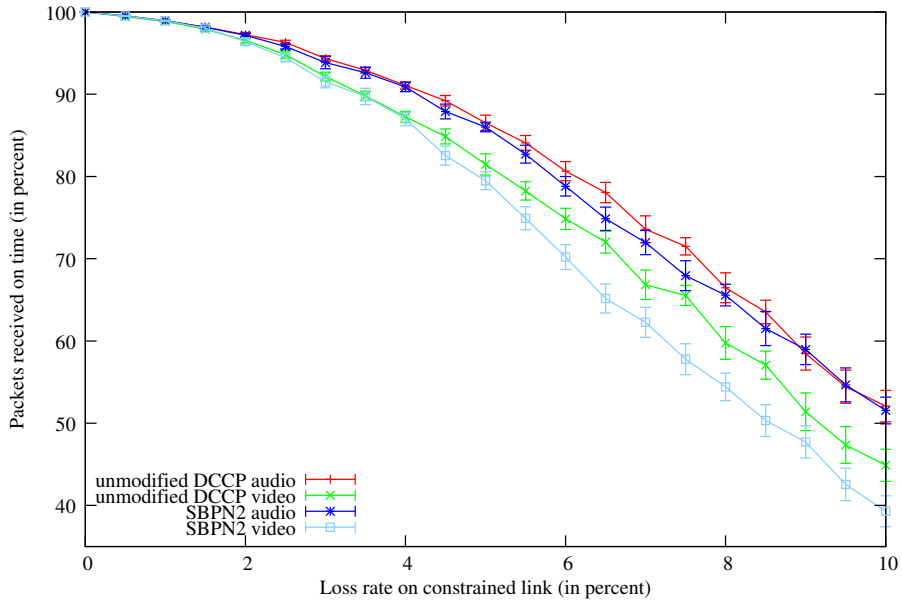


Figure 5.8: Testing with loss - 80 ms RTT, queue length 5

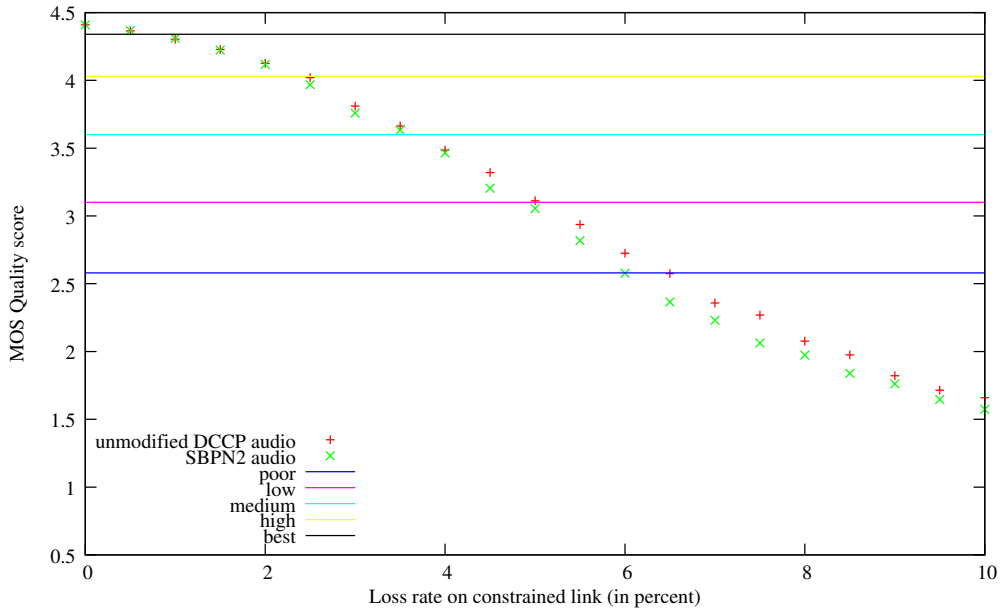


Figure 5.9: Testing with loss - 80 ms RTT, queue length 5

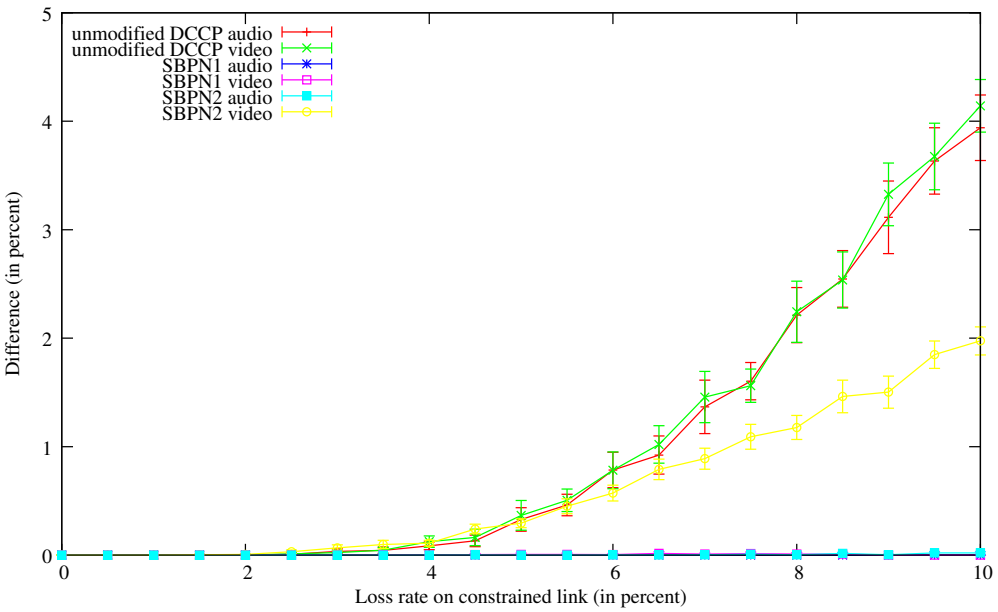


Figure 5.10: Difference in on time arrival - 80 ms RTT, queue length 5

tween the packets received and the packets received on time as it only transmits when the packet will reach the other end without expiring. SBPN2 has slightly more packets received than packets received on time as it sometimes sends data to keep the connection from going idle. Unmodified DCCP has even more packets received as it always tries to send. This indicates the improvement is not occurring because of running at a fixed rate of loss, and if fewer packets are being sent then a lesser number of relevant packets will be received due to the smaller number of packets being transmitted. It is interesting to note on this graph that all the packets used to keep alive the connection for the SBPN2 algorithm are video packets - this shows that the audio packets are being transmitted and being received on-time successfully. This can be seen from the graph as the difference between received and on-time packets for SBPN2 audio is zero, as also occurs for audio on SBPN1.

This effect can also be seen in congestion control on mobile networks where loss is not an indication of congestion but an indication of packet corruption. Congestion control algorithms such as Vegas [23] and Veno [42] attempt to determine if packet loss is due to congestion by examining the latency and whether congestion is expected. The congestion control for DCCP CCID3

that is used as the underlying congestion control model is based on TCP Reno, which does not try to determine the source of loss. It would be a useful avenue for further research to implement differing TCP based congestion control algorithms to see whether SBPN would adapt to loss better.

5.3 Testing with congestion

5.3.1 Testing SBPN2 with congestion

As tests with fixed loss (designed to mimic the effects of congestion) did not generate results for SBPN that were an improvement, tests were carried out with congestion generated directly. This was done by running tests with two competing TCP flows, produced using iperf [56] modified to allow it to run continuously. Within the video conferencing flow there are a range of larger packets for video and smaller packets for audio as described in Chapter 4, resulting in an average packet size of 519 bytes. A fixed packet size of 519 bytes was used with iperf as DCCP CCID3 works on a rate of packets per second as shown in Section 3.2.2. The link was constrained to a fixed speed by Netem for each test run, and because there are competing flows from iperf that are not at a fixed rate, congestion will occur.

The Linux kernel allows the congestion control variant for TCP to be chosen at runtime. Reno was chosen for the TCP congestion control as DCCP CCID3 uses the TCP throughput equation [84] modelled on Reno performance as discussed in the previous section. It would be interesting in future research to compare DCCP with a throughput matching the more modern TCP implementations such as BIC and Cubic in Linux, but this was not done as these had not been implemented for DCCP congestion control in Linux.

When the tests were run with competing TCP flows the results obtained are shown in Figure 5.11 and Figure 5.12. In Figure 5.11 there is a clear increase in audio packets arriving on-time with SBPN2 when compared to unmodified DCCP. At lower bit-rates more video packets arrive on-time with SBPN2 due

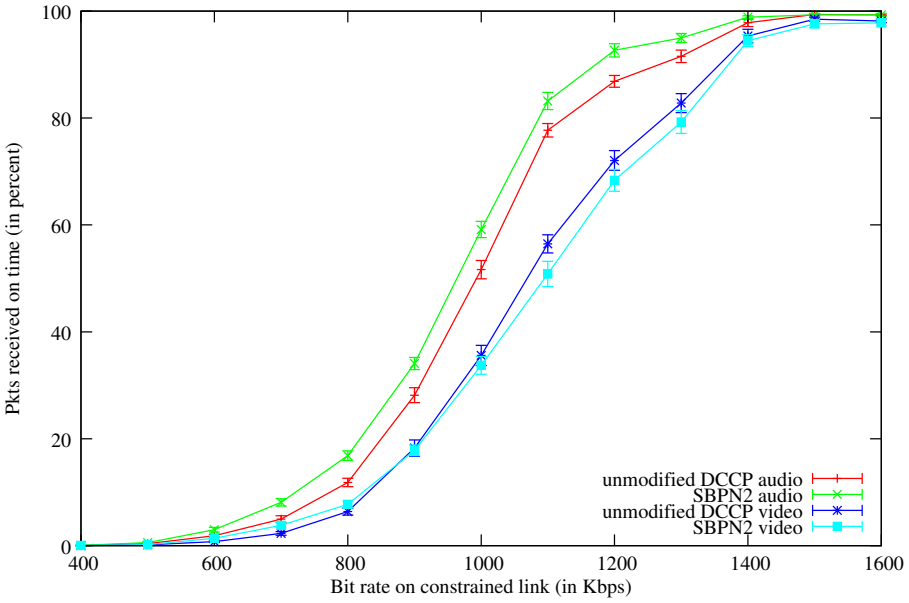


Figure 5.11: Unmodified DCCP vs SBPN2 with congestion - 80 ms RTT, queue length 5

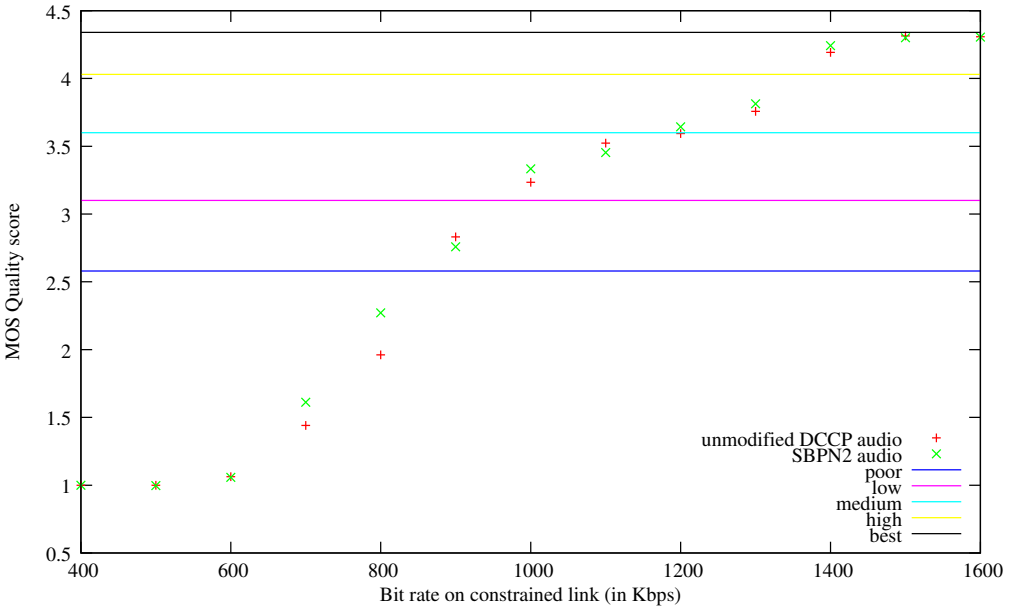


Figure 5.12: Unmodified DCCP vs SBPN2 with congestion - 80 ms RTT, queue length 5

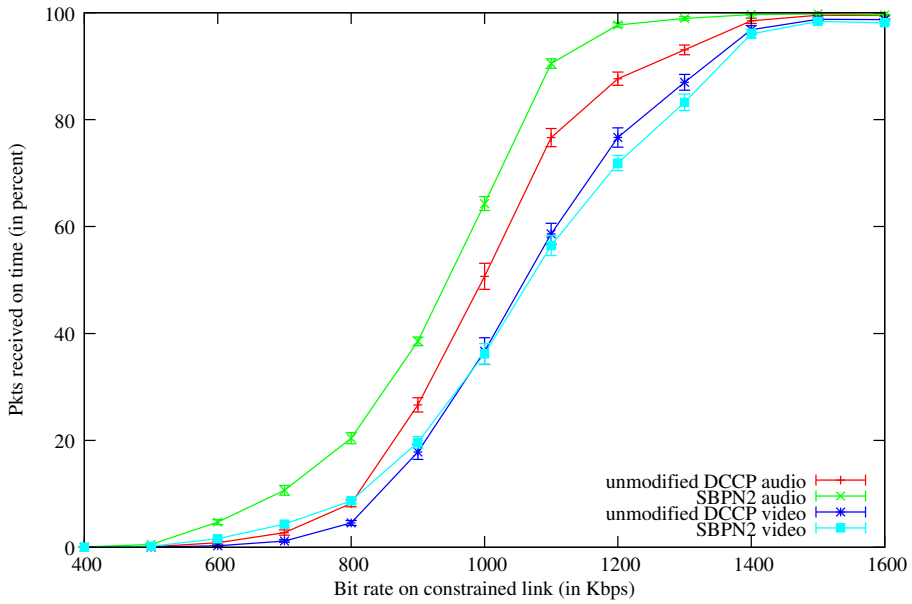


Figure 5.13: Unmodified DCCP vs SBPN2 with congestion - 80 ms RTT, queue length 32

to packet delays being long with unmodified DCCP, but as bit-rates increase fewer video packets arrive on-time with SBPN2 due to these being dropped in preference to audio packets. This result shows that more audio packets can be transmitted when there is congestion than under unmodified DCCP.

Figure 5.12 shows that quality improved for some bit-rates, especially at 700-800 Kbps. It did not improve at other bit-rates as much. The data used to produce the MOS quality score shows that this is due to higher loss rate with SBPN2, than unmodified DCCP. This again shows that the effect of discarding packets, even if they will not arrive "on time" before they reach the receiver, is detrimental to the MOS quality score.

5.3.2 Testing differing queue lengths with SBPN2

It was decided to run tests with a longer queue length to see how the SBPN2 algorithm behaved. The default TCP transmit buffer was 16 Kilobytes at the time of implementation and a queue length of 32 was therefore chosen based on an average packet size of 519 bytes from the video model constructed in Chapter 4. Figure 5.13 shows the results using a queue length of 32. These

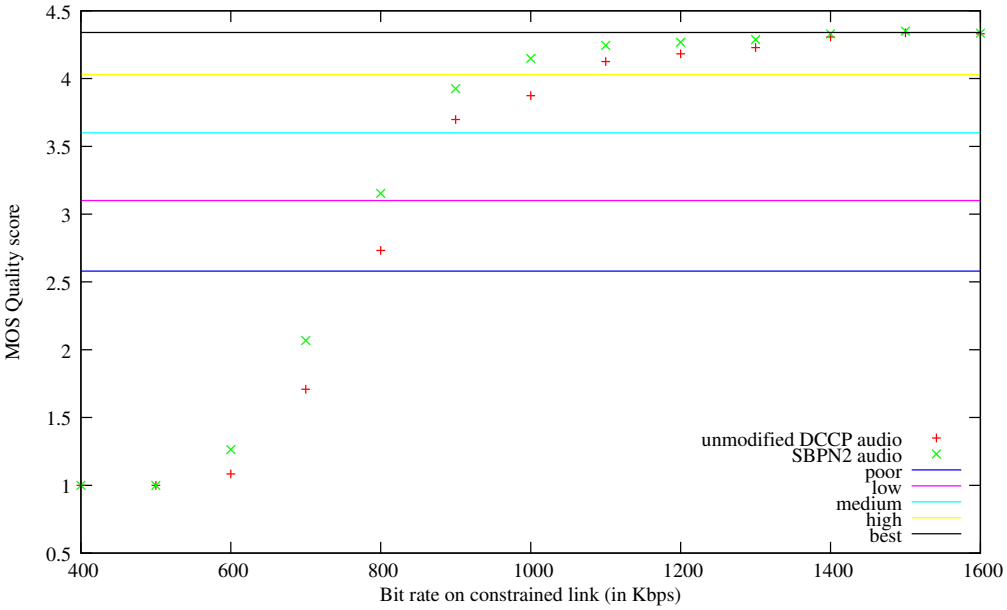


Figure 5.14: Unmodified DCCP vs SBPN2 with congestion - 80 ms RTT, queue length 32

results show a larger improvement in audio and a smaller reduction in the number of video packets dropped when compared to the results in Figure 5.11 where a queue length of 5 is used. In Figure 5.14 an increase in quality can be seen across the range with the largest improvements between 600 and 1100 Kbps. For example the MOS score improves at 800 Kbps from 2.73 to 3.15 and this lifts the quality level from poor to low.

Having a larger transmit queue means that when a packet is dropped there is less chance of there being an empty buffer, which would lead to a decreased send rate as described in Section 3.5. This is demonstrated in the results.

Figure 5.15 and Figure 5.16 show the difference when queue length is increased from 5 to 32 with unmodified DCCP. By comparison Figure 5.17 and Figure 5.18 show the difference for SBPN2 when the queue length is increased. It is interesting to note that with unmodified DCCP that there was very little difference in on-time arrival for audio when the queue length changed, where there was a difference on SBPN2. It can also be seen that the quality improves more on SBPN2 with the increase in queue length than unmodified DCCP. From Figure 5.16 and Figure 5.18 it can be seen that making queue

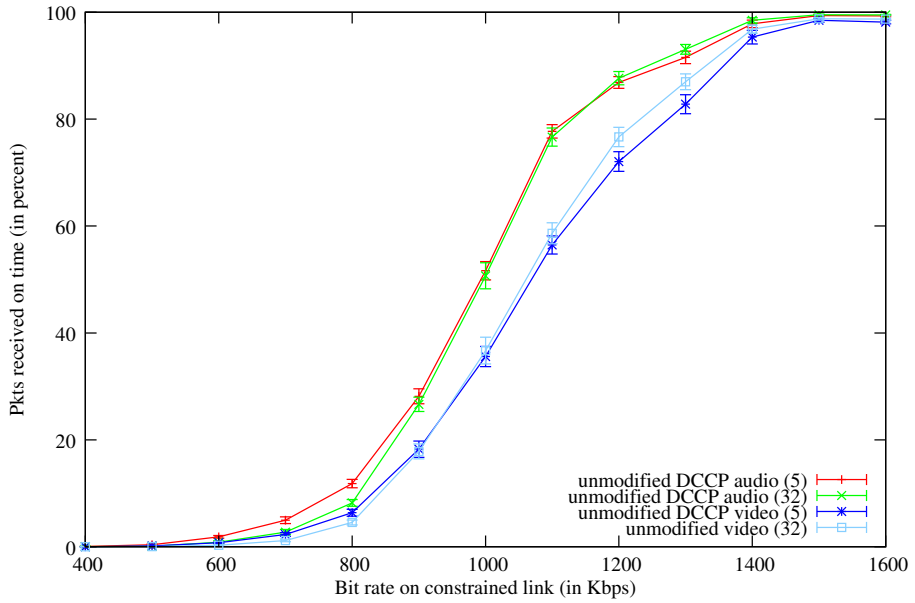


Figure 5.15: Unmodified DCCP with congestion, 80 ms RTT, queue length 5 vs 32

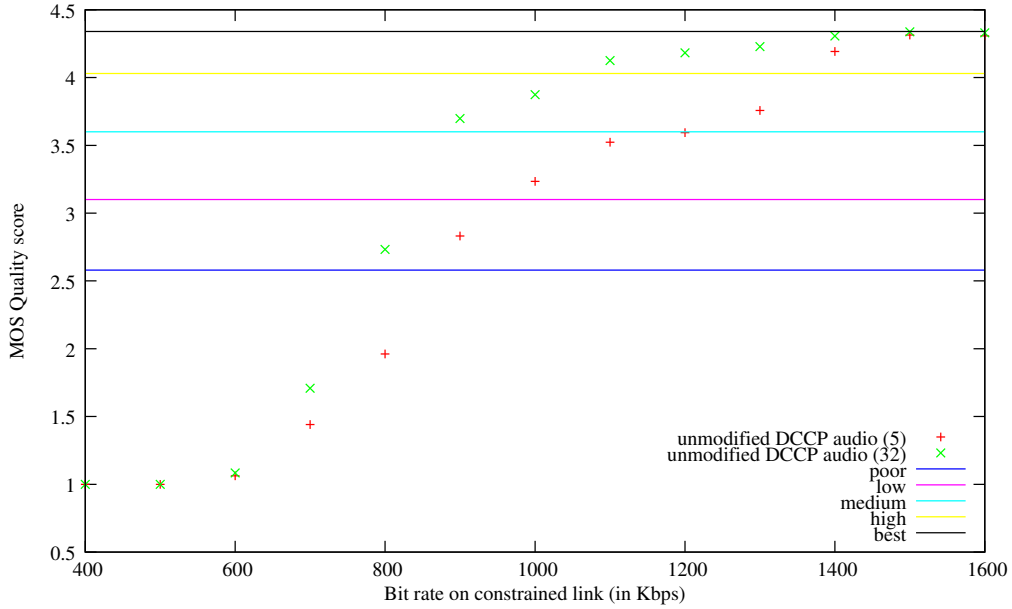


Figure 5.16: Unmodified DCCP with congestion, 80 ms RTT, queue length 5 vs 32

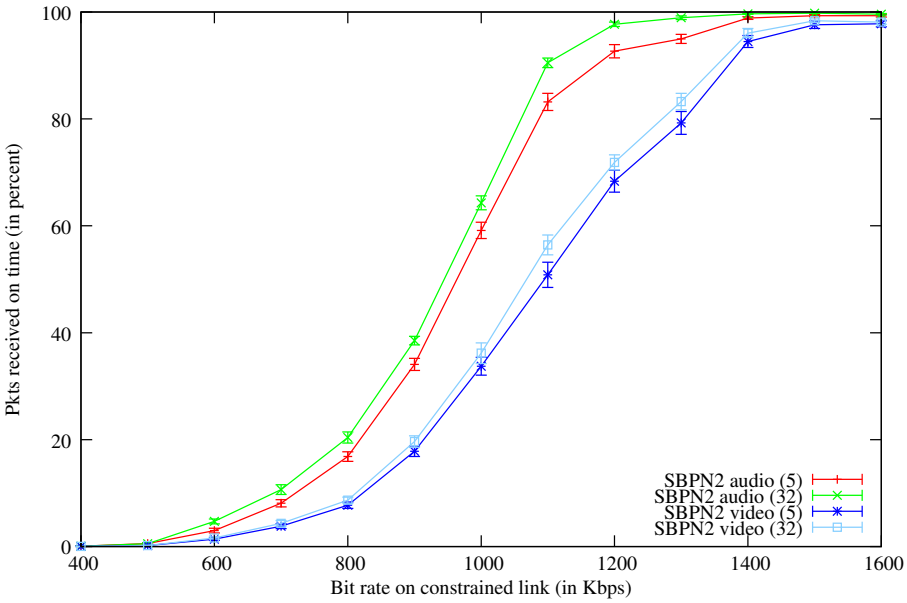


Figure 5.17: SBPN2 with congestion, 80 ms RTT, queue length 5 vs 32

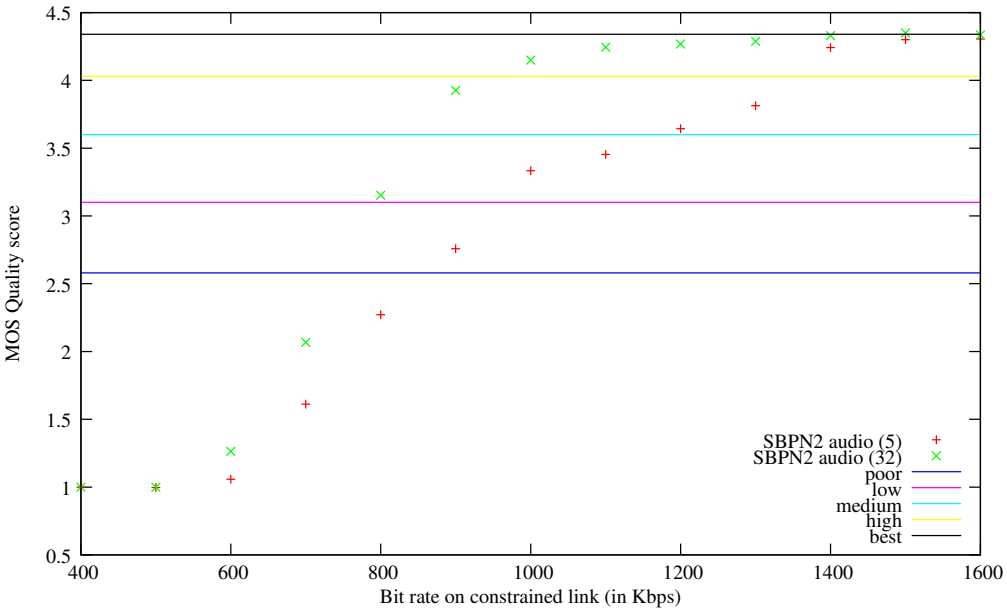


Figure 5.18: SBPN2 with congestion, 80 ms RTT, queue length 5 vs 32

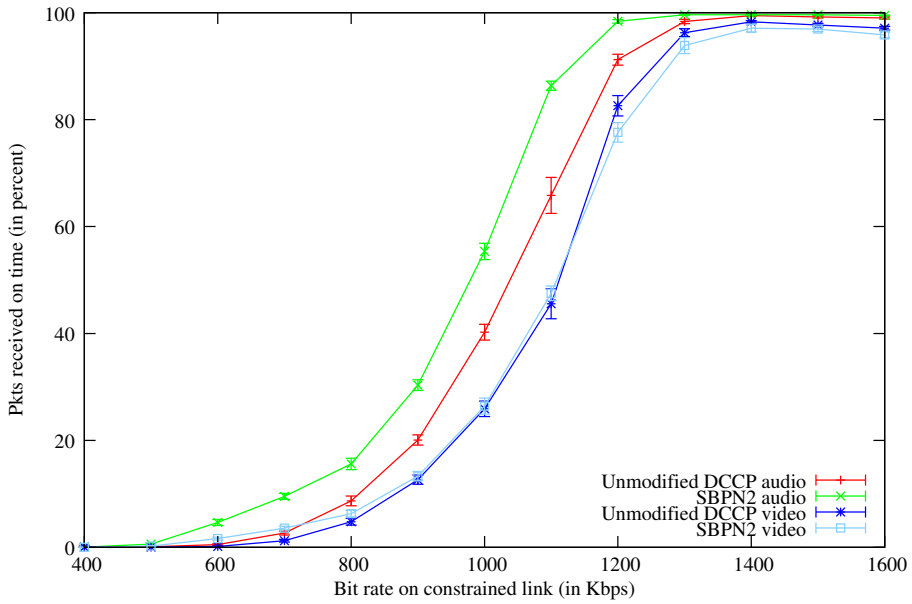


Figure 5.19: Unmodified DCCP vs SBPN2 with congestion - 30 ms RTT, queue length 32

lengths longer makes a substantial quality difference in all cases. Looking at the underlying data this is because fewer packets are discarded and this outweighs any increase in queueing time.

5.3.3 Testing SBPN2 with varying RTTs

To test if the algorithm is RTT dependent, the tests were also run with 30 millisecond and 150 millisecond RTT. The graphs for on-time delivery and quality for 30 millisecond RTT are shown in Figures 5.19 and 5.20. The graphs for 150 millisecond RTT are shown in 5.21 and 5.22. These can be compared to the 80 millisecond RTT results shown in Figure 5.13 and 5.14 earlier in this chapter. These represent scenarios such as within a continent (30 milliseconds), and between continents (150 milliseconds). These results showed that SBPN2 improved the performance of audio over a range of RTTs.

It can be seen from the graphs that the improvement does vary by the length of the RTT — quality improvement is higher for 30 milliseconds RTT than 150 milliseconds RTT. Tables 5.2, 5.3 and 5.4 show the underlying data used to calculate the quality metrics in Figures 5.19, 5.13 and 5.21. Rate is the

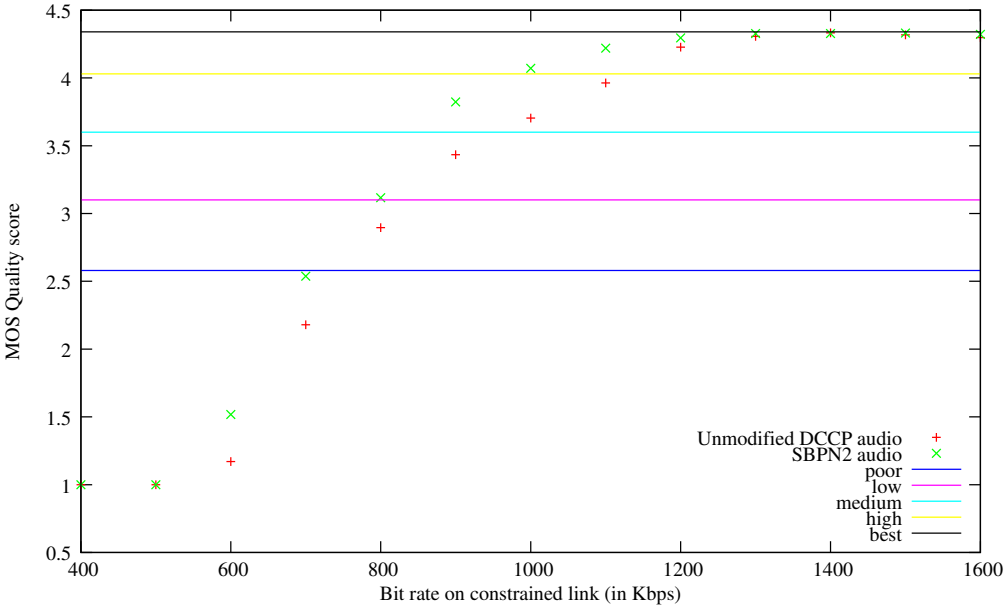


Figure 5.20: Unmodified DCCP vs SBPN2 with congestion - 30 ms RTT, queue length 32

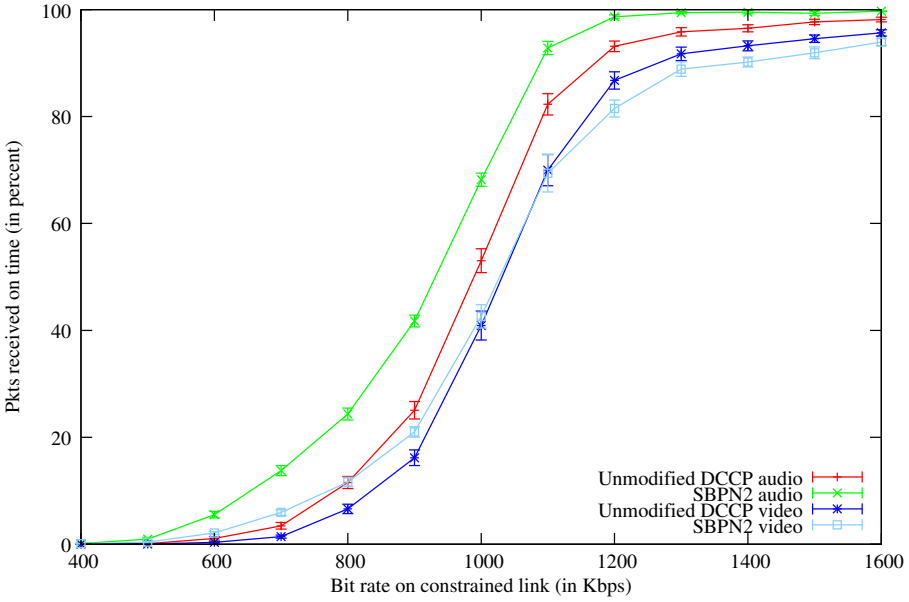


Figure 5.21: Unmodified DCCP vs SBPN2 with congestion - 150 ms RTT, queue length 32

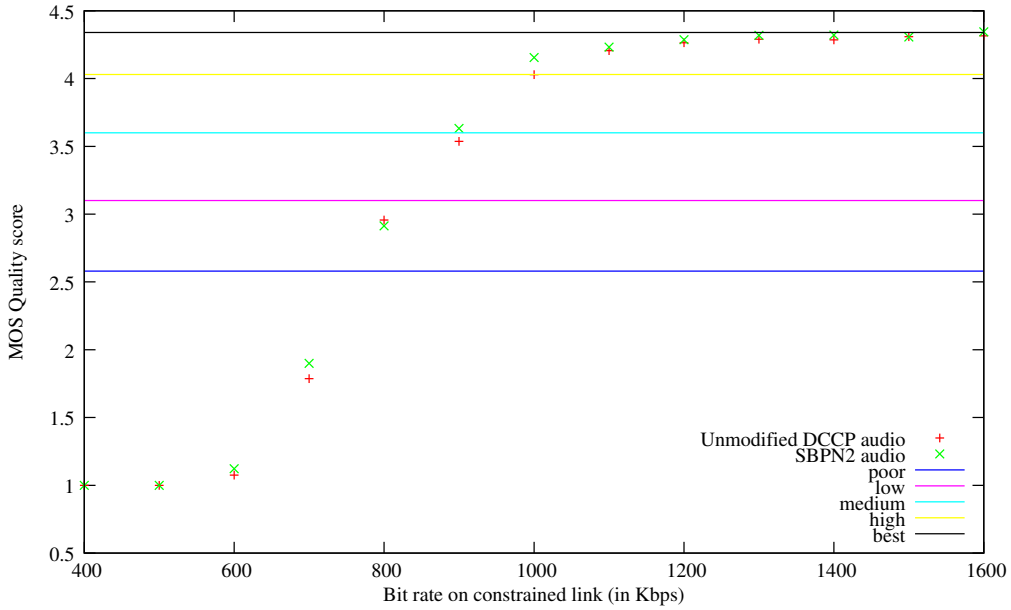


Figure 5.22: Unmodified DCCP vs SBPN2 with congestion - 150 ms RTT, queue length 32

constrained link rate in Kbits/sec, loss is the percentage of packets that did not arrive at the receiver. I_e is the impact of loss as defined in Equation 4.4, I_d is the impact of delayed packets as defined in Equation 4.5, and MOS is the Mean Opinion Score as defined in Equation 4.1.

With the rate of the constrained link between 800 and 1000 Kbps it can be seen from Tables 5.2, 5.3 and 5.4 that the improvement in quality is higher for a low RTT than for a higher RTT. This is due mostly to the variance in the impact of loss (I_e) which is derived from the loss rate. For example at 900 Kbps Table 5.2 shows the loss rate dropping from 4.19% to 2.66% for 30 millisecond RTT with unmodified DCCP compared to SBPN2, Table 5.3 shows the loss rate dropping from 2.86% to 2.28% for 80 millisecond RTT and Table 5.4 shows the loss rate *increasing* from 3.58% to 4.74% for 150 millisecond RTT.

The packet expiry time consists of the time to process a packet, queueing time and time in transit on the network as shown in Figure 4.5. The reason that SBPN2 performs better at lower RTTs is because the algorithm is less likely to discard packets as there is less probability that a packet will be likely to expire in transit due to the shorter RTT.

Table 5.2: Loss and delay with queue length 32, 30 ms RTT

	Unmodified DCCP					SBPN2				
Rate	Loss	Delay	I_e	I_d	MOS	Loss	Delay	I_e	I_d	MOS
100	88.44%	4476	79.74	580.32	1.00	98.28%	2045	82.69	254.54	1.00
200	76.30%	2164	75.64	270.43	1.00	95.54%	967	81.90	110.13	1.00
300	61.52%	1283	69.75	152.43	1.00	90.07%	637	80.25	65.89	1.00
400	46.78%	863	62.45	96.12	1.00	73.78%	480	74.71	44.75	1.00
500	30.06%	597	51.20	60.52	1.00	51.48%	372	64.98	30.34	1.00
600	17.02%	438	38.04	39.15	1.17	22.81%	309	44.60	21.88	1.52
700	9.01%	341	25.66	26.22	2.18	10.49%	269	28.36	16.54	2.54
800	6.20%	283	19.73	18.38	2.90	6.77%	241	21.03	12.85	3.12
900	4.19%	242	14.64	12.97	3.43	2.66%	213	10.08	9.08	3.82
1000	3.35%	217	12.22	9.64	3.70	1.68%	192	6.76	6.18	4.07
1100	2.44%	194	9.35	6.44	3.96	1.04%	173	4.35	4.15	4.22
1200	1.04%	164	4.34	3.94	4.23	0.48%	158	2.09	3.79	4.30
1300	0.47%	143	2.05	3.43	4.31	0.31%	136	1.38	3.26	4.33
1400	0.31%	129	1.35	3.09	4.33	0.36%	129	1.56	3.09	4.33
1500	0.50%	121	2.15	2.90	4.32	0.36%	121	1.58	2.90	4.33
1600	0.72%	114	3.06	2.74	4.30	0.50%	114	2.15	2.74	4.32
1700	0.65%	107	2.79	2.57	4.31	0.48%	108	2.10	2.59	4.33
1800	0.55%	101	2.36	2.42	4.32	0.41%	101	1.80	2.42	4.34
1900	0.52%	95	2.26	2.29	4.33	0.40%	95	1.73	2.27	4.34
2000	0.51%	92	2.23	2.22	4.33	0.38%	90	1.68	2.16	4.35

Table 5.3: Loss and delay with queue length 32, 80 ms RTT

	Unmodified DCCP					SBPN2				
Rate	Loss	Delay	I_e	I_d	MOS	Loss	Delay	I_e	I_d	MOS
100	87.70%	4467	79.50	579.08	1.00	97.91%	2000	82.58	248.56	1.00
200	71.59%	1928	73.89	238.79	1.00	93.37%	945	81.25	107.17	1.00
300	58.69%	1279	68.48	151.84	1.00	75.43%	622	75.32	63.81	1.00
400	45.59%	854	61.77	94.91	1.00	67.14%	467	72.13	43.14	1.00
500	32.56%	605	53.17	61.60	1.00	51.73%	372	65.10	30.28	1.00
600	18.90%	450	40.33	40.80	1.08	31.84%	304	52.61	21.19	1.26
700	13.24%	362	32.82	29.03	1.71	17.11%	265	38.15	15.98	2.07
800	7.16%	290	21.87	19.35	2.73	7.02%	232	21.59	11.61	3.15
900	2.86%	230	10.70	11.30	3.70	2.28%	204	8.83	7.86	3.93
1000	2.69%	204	10.16	7.80	3.87	1.37%	183	5.61	5.07	4.15
1100	1.66%	181	6.66	4.72	4.13	0.88%	166	3.71	3.98	4.24
1200	1.40%	166	5.71	3.97	4.18	0.75%	153	3.20	3.68	4.27
1300	1.10%	151	4.57	3.63	4.23	0.64%	142	2.77	3.41	4.29
1400	0.53%	132	2.31	3.17	4.31	0.34%	128	1.50	3.08	4.33
1500	0.32%	118	1.41	2.84	4.34	0.21%	117	0.94	2.81	4.35
1600	0.44%	110	1.91	2.65	4.33	0.40%	109	1.76	2.62	4.33
1700	0.42%	102	1.84	2.46	4.34	0.29%	102	1.27	2.44	4.35
1800	0.30%	94	1.33	2.26	4.35	0.27%	94	1.19	2.26	4.36
1900	0.31%	89	1.37	2.14	4.36	0.32%	89	1.41	2.13	4.35
2000	0.33%	84	1.46	2.00	4.36	0.29%	84	1.30	2.00	4.36

Table 5.4: Loss and delay with queue length 32, 150 ms RTT

	Unmodified DCCP					SBPN2				
Rate	Loss	Delay	I_e	I_d	MOS	Loss	Delay	I_e	I_d	MOS
100	87.33%	4363	79.38	565.13	1.00	97.47%	1930	82.46	239.12	1.00
200	73.63%	2045	74.66	254.46	1.00	91.27%	940	80.62	106.51	1.00
300	56.51%	1228	67.46	145.07	1.00	83.98%	625	78.30	64.23	1.00
400	45.90%	868	61.95	96.78	1.00	62.19%	459	70.05	42.01	1.00
500	33.45%	632	53.84	65.24	1.00	51.93%	367	65.21	29.67	1.00
600	19.13%	452	40.59	41.06	1.07	40.95%	297	58.98	20.27	1.12
700	12.24%	361	31.28	28.80	1.79	21.17%	256	42.88	14.75	1.90
800	5.75%	282	18.65	18.30	2.96	9.77%	225	27.08	10.70	2.91
900	3.58%	239	12.90	12.57	3.54	4.74%	200	16.12	7.32	3.63
1000	1.73%	199	6.92	7.15	4.03	1.52%	178	6.14	4.38	4.16
1100	1.16%	172	4.80	4.13	4.21	1.02%	159	4.28	3.83	4.23
1200	0.78%	153	3.32	3.68	4.26	0.63%	145	2.73	3.48	4.29
1300	0.62%	141	2.68	3.38	4.29	0.41%	133	1.80	3.20	4.32
1400	0.72%	133	3.06	3.18	4.29	0.43%	125	1.88	3.01	4.32
1500	0.55%	124	2.38	2.98	4.31	0.62%	117	2.66	2.80	4.31
1600	0.54%	117	2.33	2.80	4.32	0.28%	112	1.25	2.69	4.35
1700	0.54%	107	2.33	2.58	4.32	0.22%	107	0.98	2.57	4.35
1800	0.24%	99	1.04	2.38	4.36	0.21%	98	0.93	2.36	4.36
1900	0.17%	94	0.74	2.27	4.37	0.19%	94	0.85	2.24	4.37
2000	0.27%	91	1.21	2.19	4.36	0.17%	90	0.75	2.16	4.37

5.4 Summary

This chapter started to evaluate SBPN through the use of experimental results by measuring on-time arrival and quality through MOS.

Section 5.2 covers testing SBPN with loss and the first algorithm SBPN1 is tested in Section 5.2.1. When tested with an unbounded queue on-time arrival improved, but MOS scores were lower due to fewer packets being transmitted. When maximum queue lengths were introduced SBPN1 proved to be worse than unmodified DCCP on both on-time arrival and MOS scores.

The SBPN algorithm was modified and SBPN2 was tested in Section 5.2.2 against both SBPN1 and unmodified DCCP. SBPN2 showed improved results compared to SBPN1, but was still giving worse results than unmodified DCCP due to fewer packets being transmitted, which reduced the allowed sending rate in future. It was decided at this point to focus on testing with congestion, instead of random loss at a fixed rate.

In Section 5.3 SBPN2 was tested with two competing TCP flows to create congestion. The testing was carried out with a range of transmission queue

sizes and RTTs and demonstrated that improvements occur across a range of conditions. As an example a a substantial improvement of 0.42 in the MOS score occurs with SBPN2 compared to unmodified DCCP at 800 Kbps with an 80 millisecond RTT in Figure 5.14.

Better results were obtained in Section 5.3.2 from using higher queue lengths. This occurred in both unmodified DCCP and SBPN2. As an example the MOS score was lifted from 2.76 to 3.93 by going from a queue length of 5 to 32 using SBPN2 on a 900 Kbps link with two competing TCP flows and an 80 millisecond RTT. This is lifting the quality from partway between poor and low, to nearly high.

Section 5.3.3 showed that SBPN2 works better on short RTT. This is because there is less probability of a packet needing to be discarded as expired.

The improvements from SBPN1 to SBPN2, increasing queue lengths and shorter RTT all demonstrate the effect loss has on the MOS quality measure. Packets being dropped by SBPN reduce the allowed sending rate and thus increase the number of packets that don't arrive on-time or at all.

The results from this chapter also show that reducing transmit rate due to network congestion or loss may not be the best method for real time media applications and this is discussed further in Chapter 8.

In the following chapters further variants in algorithms are tested to see if further improvements in audio quality can be made.

Chapter 6

Faster Restart and LIFO

6.1 Introduction

In the previous chapter SBPN was tested and improvements to quality were seen from the SBPN algorithm when testing was done with congestion. In this chapter two different ways to potentially further improve quality are described and tested. These are TFRC Faster Restart and the use of LIFO queues.

In Section 6.2 TFRC Faster Restart is introduced and tested. Faster Restart looks to increase the speed of recovery to the allowable speed after loss has been incurred on networks.

LIFO queues are introduced in Section 6.3 and tested to compare them to SBPN2. The use of priority queues with LIFO is examined also in this section and compared to queues without priority.

6.2 TFRC Faster Restart

It was shown in Section 5.2.3 that fewer packets being transmitted was resulting in lower quality due to DCCP reducing the allowed sending rate when the SBPN algorithm did not send expired packets. TFRC Faster Restart [64] seeks to improve the performance of TFRC by recovering faster from data limited transmission rates and idle periods by increasing the allowed sending rate quicker than standard TFRC. As such it was decided to test TFRC Faster

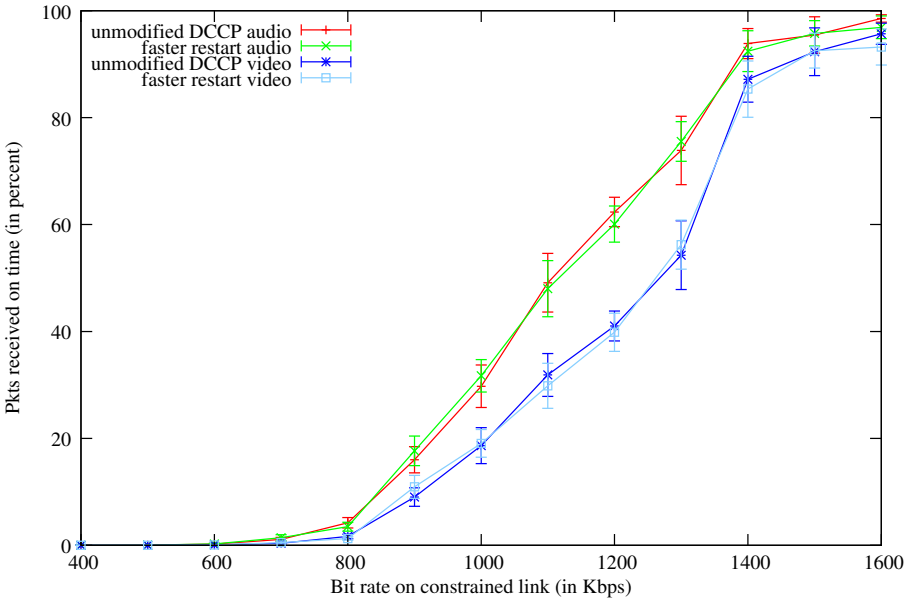


Figure 6.1: Unmodified DCCP vs Faster Restart - 80 ms RTT, queue length 5

Restart and see if it made a noticeable improvement in the video conference model.

6.2.1 Implementation of TFRC Faster Restart

Draft 3 of TFRC Faster Restart was implemented to see if this would counteract the effect of reduced transmit rates and tests were carried out. The implementation was done on Linux 2.6.23 so this graph cannot be compared directly to the other graphs in this thesis. It was implemented on this version of the Linux kernel because Faster Restart is based on a later version of DCCP CCID3 which was merged into the Linux kernel subsequent to the work done in Linux 2.6.20 which is used for all other tests.

6.2.2 Results from TFRC Faster Restart

The runs for Linux 2.6.23 showed much higher variance between runs than on other tests with Linux 2.6.20. Tests were carried out and this was found to a problem with the DCCP CCID3 code whereby it was not controlling the rate of transmission. This was due to a fault introduced when improvements were being made in another area of the CCID3 code.

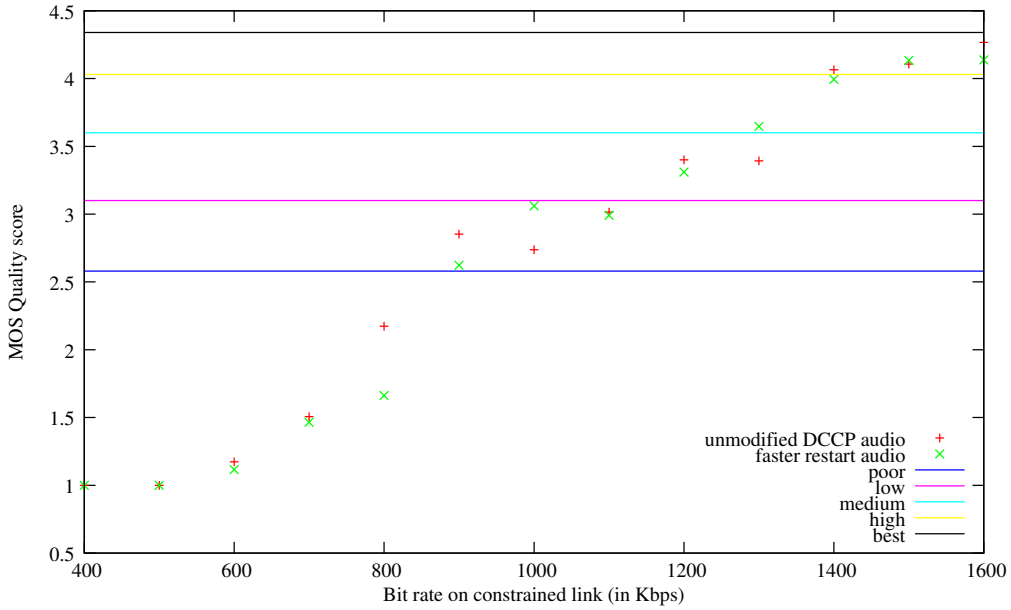


Figure 6.2: Unmodified DCCP vs Faster Restart - 80 ms RTT, queue length 5

The results for TFRC Faster Restart compared to unmodified DCCP (both on Linux 2.6.23) are shown in Figure 6.1 and Figure 6.2. The tests were conducted with fixed latency, and two competing TCP flows as described in Section 5.3.1. Figure 6.1 shows that TFRC Faster Restart did not produce significantly different results to unmodified DCCP when measuring packets received on time. The results from Figure 6.2 do show difference in results at 800, 900, 1000 and 1300 Kbps. These results however are not consistent with each other and are probably due to the high variability in transmission rate.

Analysing both the results and the protocol it is shown that TFRC Faster Restart does not offer improvements in the scenario where there is both loss occurring and the application sending rate is limited by DCCP. The limiting factor with DCCP is the allowed sending rate being lowered due to loss, rather than being able to recover from idle periods more quickly.

This lack of improvement in TFRC Faster Restart under the test scenario was also discussed with Arjuna Sathianseelan from the University of Aberdeen who was also examining TFRC Faster Restart and he also saw that Faster Restart did not give expected improvements. These results were later published in Sathianseelan and Fairhurst [101]. They saw that the allowed sending

rate from TFRC dropped quickly after small idle periods, and that this was the greatest reason for poor performance. Faster restart did not give much of an improvement as allowed sending rates had dropped so quickly — in many cases right back to the minimum initial transmit rate. Sathianseelan said that application developers responded by sending "padding" packets as we do with SBPN2. A newer version of TFRC [48] gave improvement in their research and we mention this in possibilities for future work in Chapter 8.4.

In consequence of these results, no attempt was made to back port the changes to Linux 2.6.20, or to have transmission rate stability in Linux 2.6.23 fixed, as TFRC Faster Restart did not show any potential for major improvements.

6.3 LIFO

It was decided to investigate the use of Last In First Out (LIFO) queues to see if this would provide any benefit from having the most recent packet transmitted first. LIFO queues are also known as stack,s With LIFO queues the last packet to be added is the first to be transmitted. Two variations using LIFO were implemented — LIFO priority queues and LIFO queues without priority. In LIFO priority queues audio packets are placed in the queue before video packets. In both queue types packets are still discarded if the packet will expire before it reaches the destination, as with SBPN.

Results from these algorithms are shown in Figure 6.3 and Figure 6.4 for LIFO priority queues compared to SBPN2, Figure 6.5 and Figure 6.6 for LIFO queues compared to SBPN2. The tests were conducted with fixed latency, and two competing TCP flows as described in Section 5.3.1. The two algorithms show only small differences when compared to SBPN2. It appears overall that LIFO priority queues give around the same performance as SBPN2, and LIFO queues without prority perform slightly worse although it should be noted that the difference is not significant across the whole range.

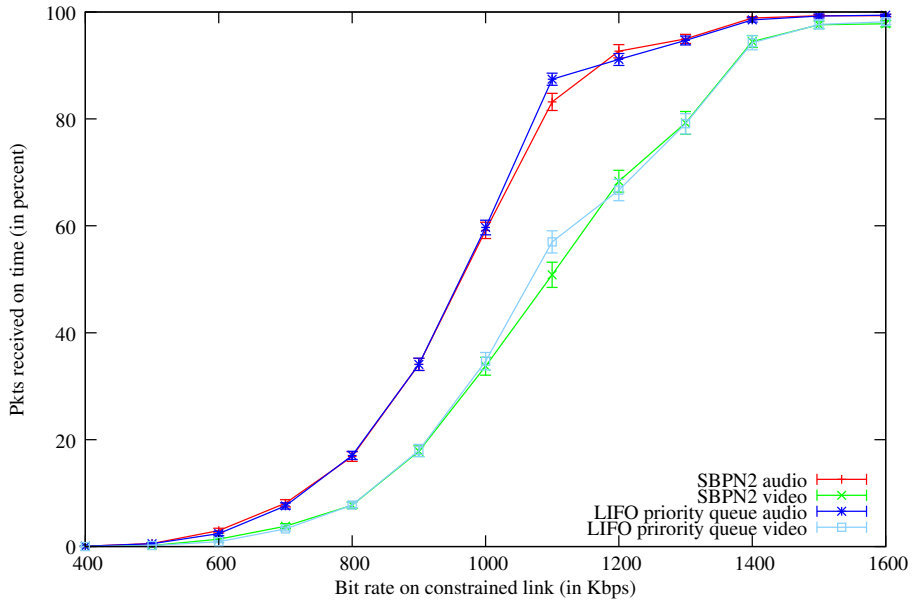


Figure 6.3: SBPN2 vs LIFO priority queues - 80 ms RTT, queue length 5

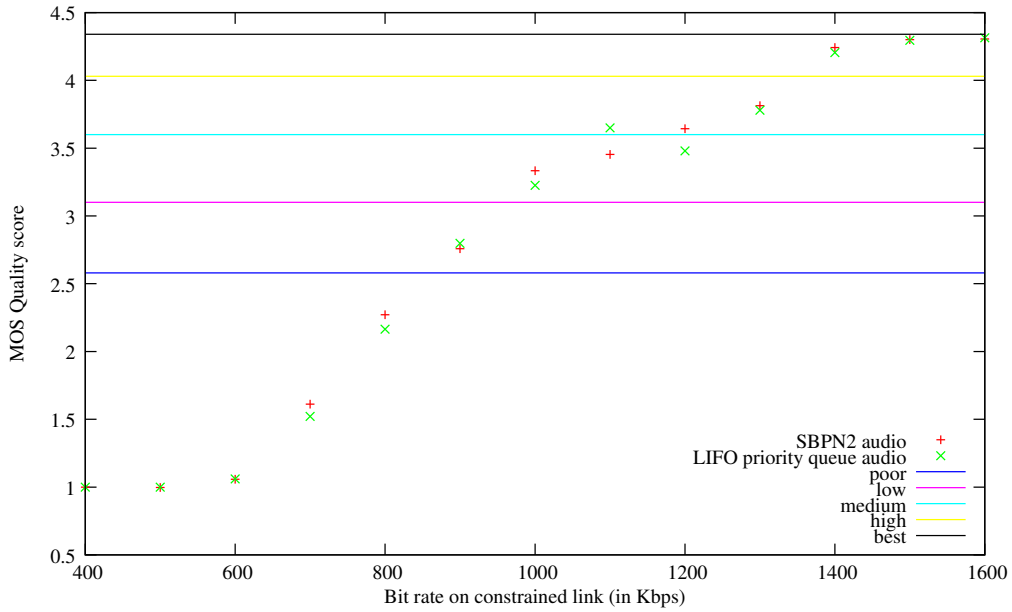


Figure 6.4: SBPN2 vs LIFO priority queues - 80 ms RTT, queue length 5

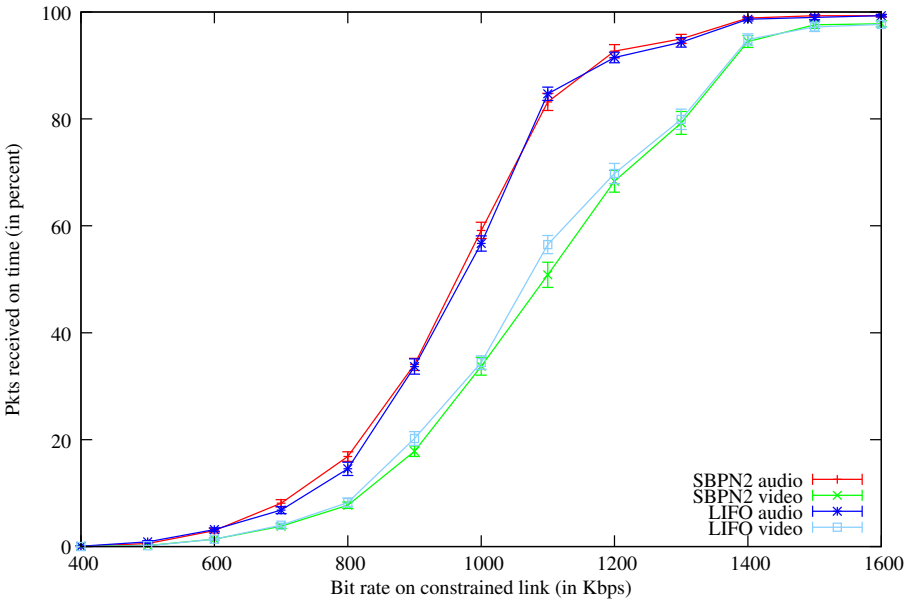


Figure 6.5: SBPN2 vs LIFO queue - 80 ms RTT, queue length 5

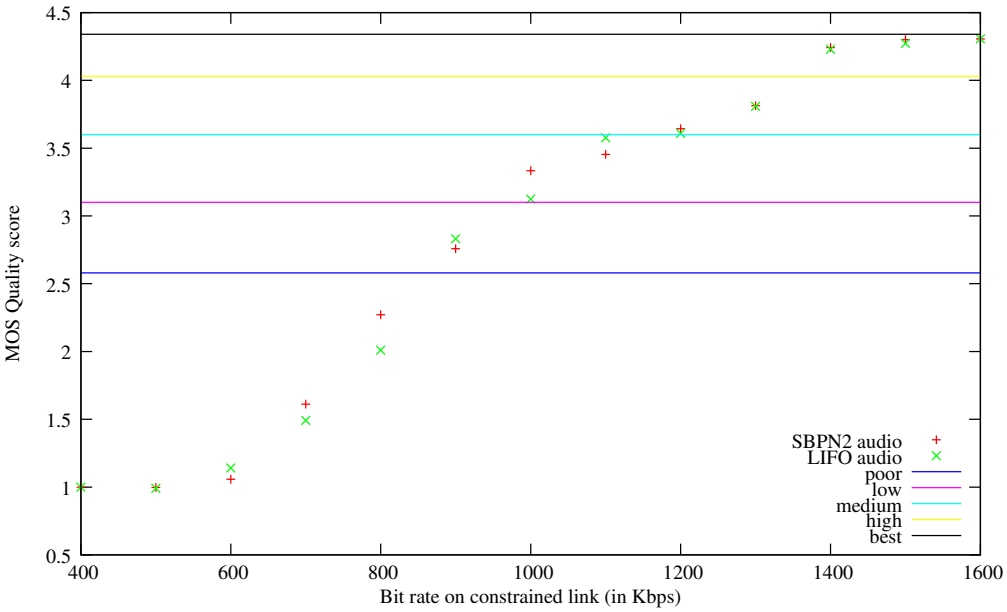


Figure 6.6: SBPN2 vs LIFO queue - 80 ms RTT, queue length 5

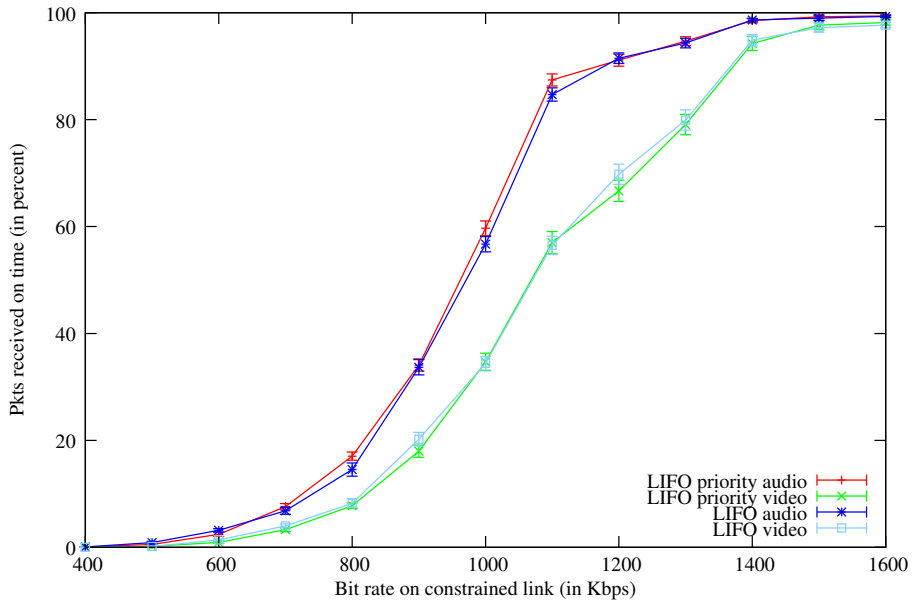


Figure 6.7: LIFO priority queues vs LIFO queue - 80 ms RTT, queue length 5

Figure 6.7 compares LIFO priority queues to a single LIFO queue. This shows that for on-time arrival for audio that LIFO prority queues perform slightly better than a non-prioritised queue. This shows that sending audio packets before video packets makes a difference in on-time arrival, which confirms what would be expected when looking at the idea. These findings also back the use of priority queues in SBPN2.

It should also be noted that if a LIFO queue is used then longer playback buffers would be needed due to packets being sent in reverse order and thus making additional delays likely. If there are additional delays this would reduce the quality of the audio and this may negate any benefits obtained from using a LIFO queue. Given that LIFO queues did not perform better than SBPN2 without quantifying these delays no attempt was made to estimate these effects. With there not being an improvement in performance the use of LIFO queues was not investigated further.

6.4 Summary

In this chapter two different techniques: Faster Restart and LIFO queues, were used to see if either could make an improvement to DCCP performance for audio quality. Neither of the two methods showed significant improvement and further testing was not carried out.

Faster Restart was designed to recover from loss quickly but showed no discernible improvement under test, mirroring other researcher's tests. LIFO queues transmitted the last packet first and although the tests did not show improvement they did however show the benefit of using priority queues which are a fundamental part of SBPN2.

In the following chapter another method of altering transmission will be investigated — ring buffers.

Chapter 7

Ring buffers

7.1 Introduction

In Chapter 5 and Chapter 6 SBPN, and variants on it, are tested and the results are analysed. This chapter continues testing variants on SBPN and contrasting different algorithms. A transmission scheme called ring buffers proposed by Kohler and Lai is discussed in Section 7.2 and a variant of this is implemented. Ring buffers alter the sending policy at time of queueing, where SBPN alters the sending policy at the time of transmission.

Section 7.3 compares SBPN2 to ring buffers with differing queue sizes and RTTs. The results obtained are discussed and contrasted to results obtained in previous chapters.

SBPN2 and ring buffer algorithms are combined in Section 7.4 to see what further improvements can be made by using both algorithms together. Data is presented from these tests and analysed to understand possible explanations for the results obtained.

Further tests are carried out in Section 7.5 to compare ring buffers with combined SBPN2 and ring buffers. It is examined what effect the RTT has on results and explanations for those results.

7.2 Ring buffers theory

A ring buffer is a data structure commonly used for implementing finite queues. Head and tail pointers into an array of storage locations track current queue contents. Ring buffers have a natural way of dealing with overflow. When the buffer is full it is easy to have a new entry overwrite the old head — discarding the oldest entry in the queue.

Kohler and Lai's [69] detail an algorithm that uses ring buffers, as described in Section 2.4.2. Their video test application generates one packet every 10, 20, 25 or 30 milliseconds which are marked as expired after three packets are queued. Their network was constrained to 50 Kilobits per second. They then also implemented preferential packet dropping; that is, if the buffer of three packets is full then their algorithm drops B and P frames before I frames so that as many I frames as possible are sent - they experimented with 10, 20, and 25 milliseconds but keeping at a size of three packets buffer.

Kohler and Lai said that using ring buffers would mean that stale packets were discarded before being transmitted and thus more recent packets would be transmitted. Their idea was implemented in this project to see if further improvements to audio quality could be made. Their exact algorithm was not implemented, but the underlying concept of removing stale packets at time of insertion into the queue by the application was implemented. The implementation of the algorithm is still called ring buffers for simplicity in this thesis.

In our implementation of ring buffers a variable length buffer was used, compared with a length of three that Kohler and Lai use. If the buffer is full at the time of queueing the packet for transmission and there are video packets in the queue, the oldest video packet is dropped, otherwise the oldest audio packet is dropped. The data is stored in a priority queue with audio packets queued before video packets. The algorithm is represented in the pseudo code shown in Figure 7.1. The logic at time of transmission is simple — send the first packet in the priority queue.

```
if buffer is not full:
  if packet is audio:
    insert at end of audio packets in priority queue
  else packet is video:
    insert at end of priority queue
else buffer is full:
  if packet is audio:
    if video packets in queue:
      discard oldest video packet
    else discard oldest audio packet
    insert at end of audio packets in priority queue
  else packet is video:
    if video packets in queue:
       discard oldest video picket
       insert at end of priority queue
    else throw away new video packet
```

Figure 7.1: Ring buffer pseudo code - insertion

SBPN2 determines whether to discard a packet at time of transmission. Kohler and Lai's ring buffers ask the application to determine whether to discard a packet at time of queuing for transmission. Our implementation was also done with the decision made on which packet to drop being made at the time of insertion into the transmit queue. But the drop decision is made by the DCCP stack, rather than the application. This can be done as the priority and expiry time is passed to the DCCP stack, enabling the same decisions to be made as would have been made by an application. In contrast to SBPN only the priority is used for ring buffers, and not the expiry time.

7.3 Results of Ring Buffer compared to SBPN2

Figures 7.2 and 7.3 compare the results for SBPN2 with the results for ring buffers when using a queue length of 5. The tests were conducted with fixed latency, and two competing TCP flows as described in Section 5.3.1. When looking at on time packet arrival in Figure 7.2 ring buffers performed slightly better than SBPN2. When looking at audio quality as shown in Figure 7.3 ring buffers gave substantially better MOS quality scores than SBPN2.

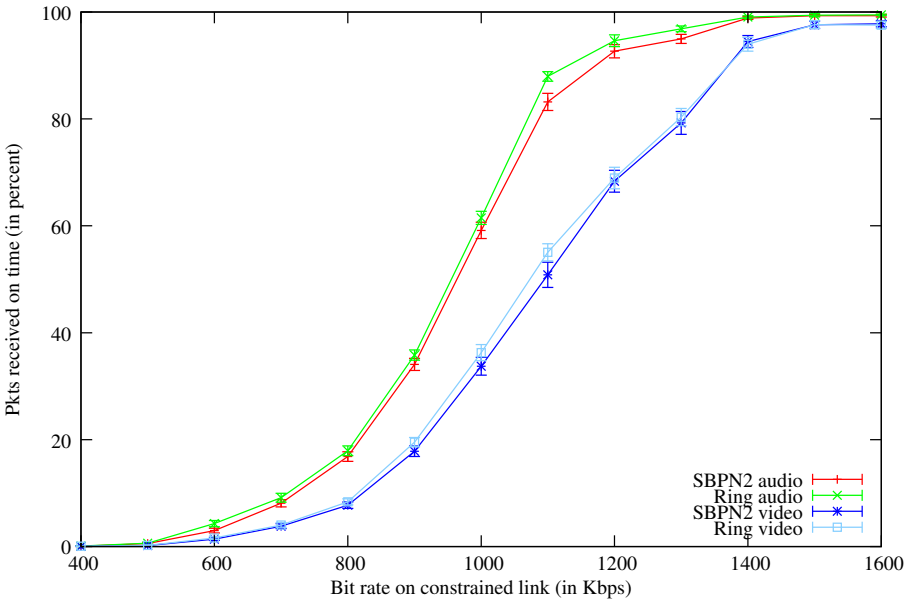


Figure 7.2: SBPN2 vs Ring buffers - 80 ms RTT, queue length 5

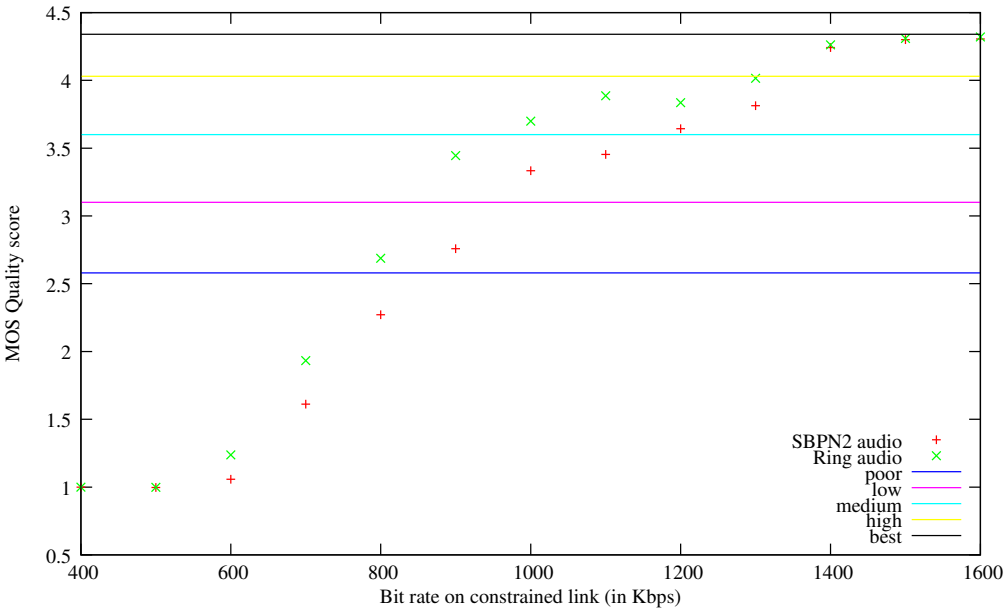


Figure 7.3: SBPN2 vs Ring buffers - 80 ms RTT, queue length 5

Table 7.1: Loss and delay with queue length 5, 80 ms RTT

	SBPN2					Ring buffers				
Rate	Loss	Delay	I_e	I_d	MOS	Loss	Delay	I_e	I_d	MOS
100	97.69%	2016	82.52	250.70	1.00	97.96%	1993	82.60	247.49	1.00
200	93.78%	943	81.37	106.88	1.00	92.80%	954	81.08	108.38	1.00
300	72.24%	619	74.14	63.45	1.00	77.29%	623	75.99	63.96	1.00
400	63.45%	468	70.59	43.26	1.00	66.99%	469	72.07	43.39	1.00
500	48.46%	373	63.38	30.41	1.00	48.32%	374	63.30	30.67	1.00
600	44.09%	308	60.90	21.73	1.06	32.54%	306	53.15	21.56	1.24
700	26.28%	266	47.94	16.19	1.61	19.26%	266	40.75	16.18	1.93
800	17.18%	234	38.24	11.82	2.27	11.55%	234	30.15	11.90	2.69
900	13.22%	205	32.79	7.93	2.76	6.06%	205	19.40	7.98	3.44
1000	8.43%	184	24.52	5.09	3.33	5.05%	183	16.91	5.07	3.70
1100	7.77%	167	23.19	4.00	3.45	3.85%	166	13.69	3.99	3.89
1200	6.12%	152	19.53	3.66	3.64	4.39%	153	15.18	3.68	3.84
1300	4.70%	142	16.00	3.41	3.81	2.96%	142	11.02	3.41	4.02
1400	1.12%	129	4.66	3.10	4.24	0.95%	129	3.99	3.10	4.26
1500	0.68%	117	2.90	2.81	4.30	0.60%	117	2.57	2.82	4.31
1600	0.67%	110	2.88	2.65	4.31	0.53%	111	2.30	2.65	4.32
1700	0.61%	102	2.64	2.45	4.32	0.37%	102	1.62	2.45	4.34
1800	0.81%	95	3.45	2.29	4.30	0.40%	95	1.74	2.29	4.34
1900	0.42%	89	1.85	2.14	4.34	0.35%	90	1.55	2.16	4.35
2000	0.42%	84	1.81	2.02	4.35	0.34%	84	1.50	2.02	4.36

The data that is used to calculate the audio quality is presented in Table 7.1. The explanations of I_e, I_d and MOS are given in Section 5.2.1.1.

This table shows that the packet loss rate for SBPN2 is higher than the packet loss rate for ring buffers, while the packet delay is around the same for both algorithms. The packet loss rate being higher for SBPN was a problem that was observed in Section 5.2.3 and is due to DCCP CCID3 slowing down the allowable transmission rate if no packets are being transmitted.

When the comparison was done with a queue length of 32 as shown in Figure 7.4 and Figure 7.5 there was no significant difference in behaviour between SBPN2 and ring buffers — both when measured by on time arrival and by quality. When looking at the measures used in the MOS quality score as shown in Table 7.2 it can be seen that, unlike with queue length 5, the loss rate is almost the same across both algorithms. This is because with larger buffers the probability of buffers running out of packets to transmit are much lower, and thus the transmission rate is not reduced and therefore reducing the loss rate.

The tests were also repeated with RTTs of 30 milliseconds and 150 mil-

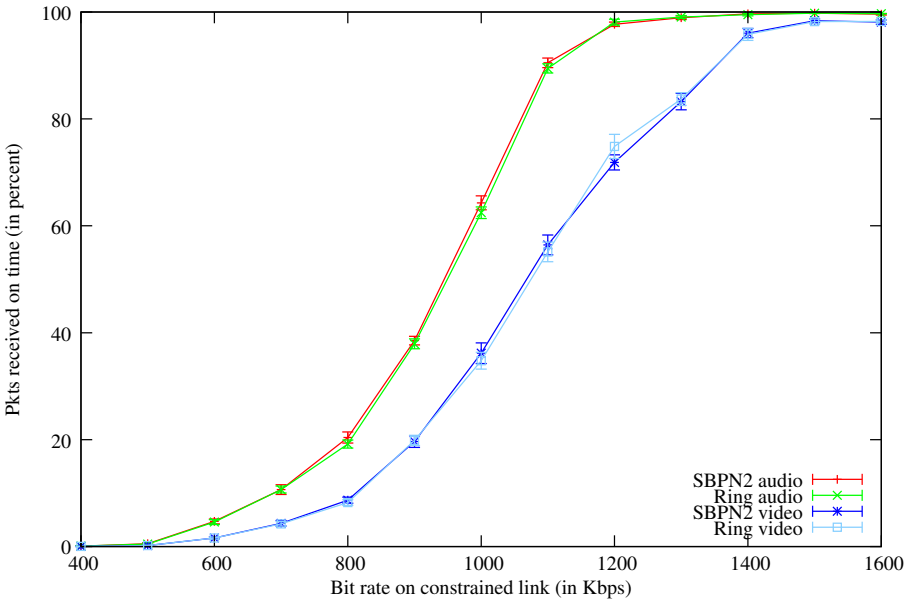


Figure 7.4: SBPN2 vs Ring buffers - 80 ms RTT, queue length 32

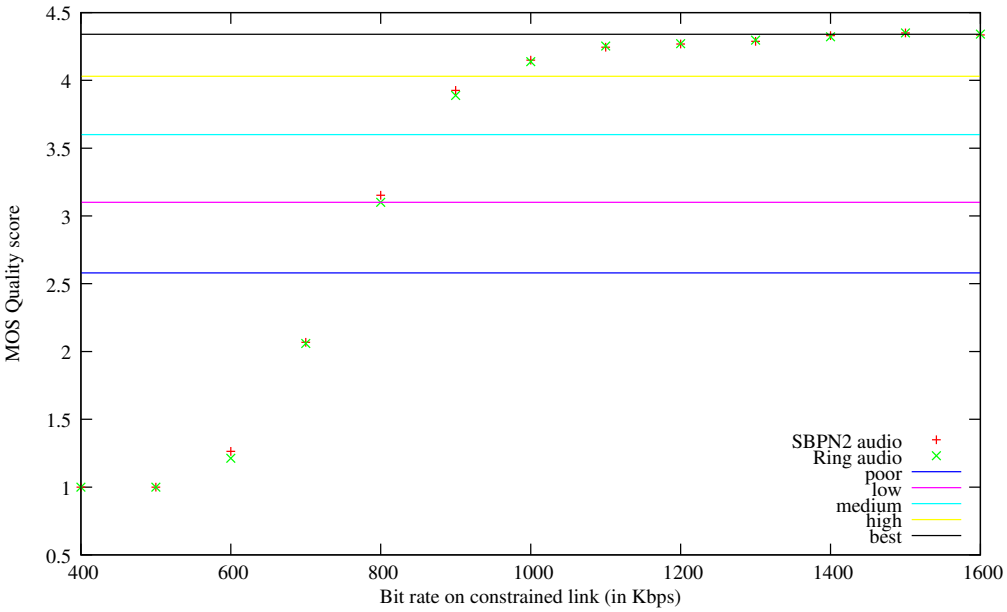


Figure 7.5: SBPN2 vs Ring buffers - 80 ms RTT, queue length 32

Table 7.2: Loss and delay with queue length 32, 80 ms RTT

	SBPN2					Ring buffers				
Rate	Loss	Delay	I_e	I_d	MOS	Loss	Delay	I_e	I_d	MOS
100	97.91%	2000	82.58	248.56	1.00	97.38%	1971	82.43	244.62	1.00
200	93.37%	945	81.25	107.17	1.00	93.18%	925	81.19	104.45	1.00
300	75.43%	622	75.32	63.81	1.00	76.60%	631	75.75	65.09	1.00
400	67.14%	467	72.13	43.14	1.00	58.44%	469	68.37	43.32	1.00
500	51.73%	372	65.10	30.28	1.00	50.71%	376	64.58	30.84	1.00
600	31.84%	304	52.61	21.19	1.26	33.90%	305	54.18	21.42	1.21
700	17.11%	265	38.15	15.98	2.07	17.14%	266	38.19	16.10	2.06
800	7.02%	232	21.59	11.61	3.15	7.33%	235	22.25	11.93	3.10
900	2.28%	204	8.83	7.86	3.93	2.53%	205	9.64	7.95	3.89
1000	1.37%	183	5.61	5.07	4.15	1.42%	185	5.79	5.24	4.14
1100	0.88%	166	3.71	3.98	4.24	0.79%	167	3.37	4.00	4.25
1200	0.75%	153	3.20	3.68	4.27	0.76%	148	3.24	3.55	4.27
1300	0.64%	142	2.77	3.41	4.29	0.56%	142	2.44	3.40	4.30
1400	0.34%	128	1.50	3.08	4.33	0.42%	129	1.82	3.10	4.32
1500	0.21%	117	0.94	2.81	4.35	0.21%	118	0.94	2.82	4.35
1600	0.40%	109	1.76	2.62	4.33	0.33%	110	1.43	2.65	4.34
1700	0.29%	102	1.27	2.44	4.35	0.39%	102	1.72	2.45	4.34
1800	0.27%	94	1.19	2.26	4.36	0.30%	96	1.31	2.29	4.35
1900	0.32%	89	1.41	2.13	4.35	0.31%	90	1.38	2.16	4.35
2000	0.29%	84	1.30	2.00	4.36	0.27%	85	1.20	2.04	4.36

liseconds and a queue length of 32. With these tests there was no significant difference in the quality results between SBPN2 and ring buffers.

In Figure 7.6 and Figure 7.7 the results of varying the ring buffer queue between length of 5 and 32 are directly compared. This shows that increasing the size of the queue length does improve the quality with ring buffers. This compares with Figure 5.18 where SBPN2 shows a bigger increase in quality with the queue length increasing from 5 to 32. The reason for ring buffer doing better with shorter queues is that the ring buffer algorithm does not discard packets if the packet is old and the loss rate is lower, and thus the transmission rate is not decreased.

7.4 Combining SBPN2 and Ring buffers

SBPN2 determines whether to discard a packet at time of transmission. Ring buffers determine whether to discard a packet at time of queueing for transmission. It was decided to combine the two approaches in one algorithm for comparison to see if the combination further improves the number of packets

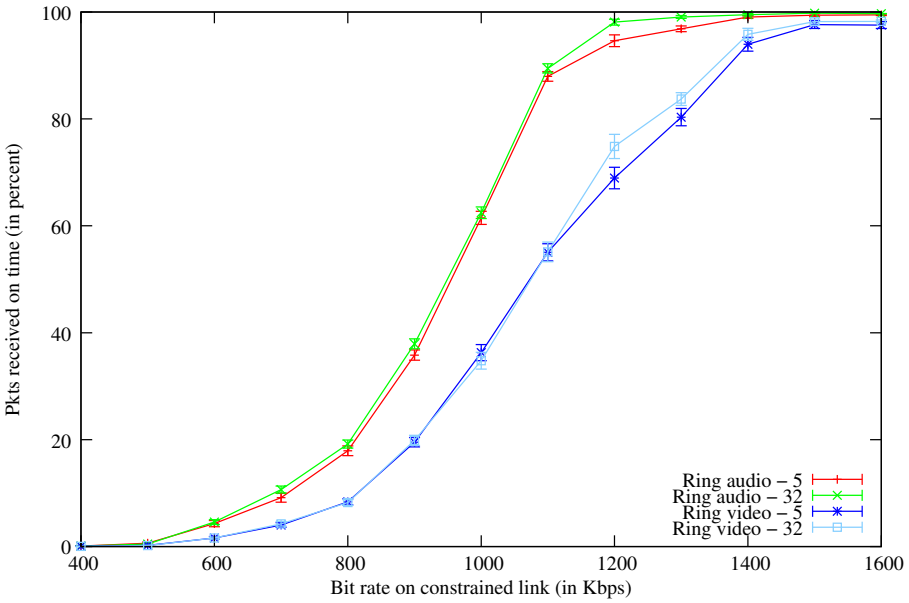


Figure 7.6: Ring buffers - 80 ms RTT, queue length 5 vs 32

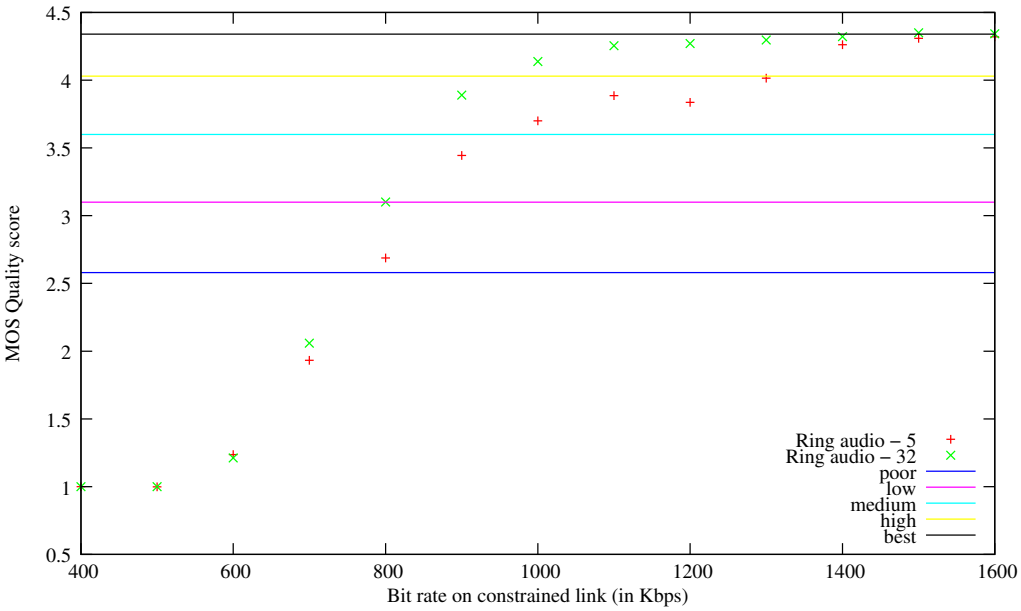


Figure 7.7: Ring buffers - 80 ms RTT, queue length 5 vs 32

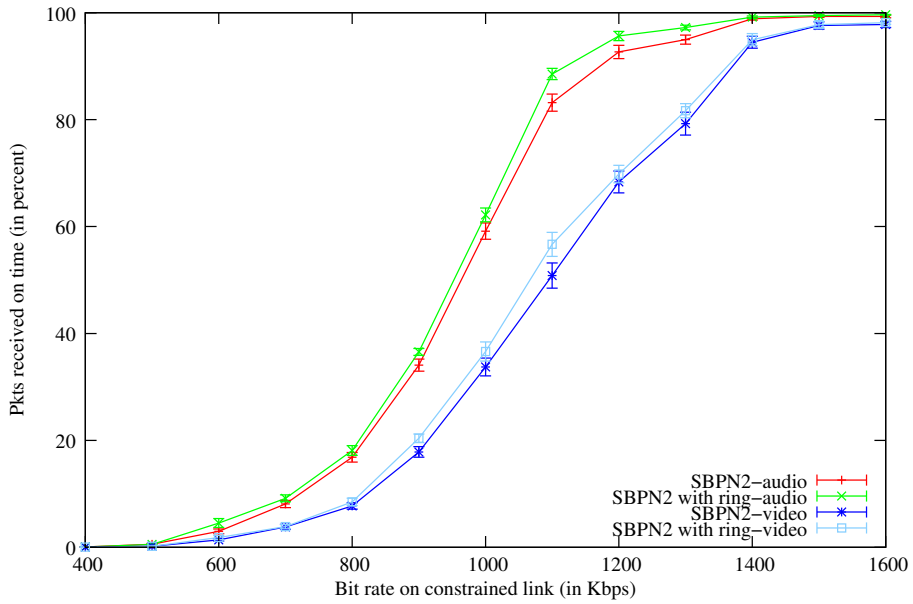


Figure 7.8: SBPN2 vs SBPN2 with ring - 80 ms RTT, queue length 5

received on time.

The algorithm for SBPN2 with ring buffers is the same as ring buffers for packet insertion as described in Figure 7.1. At the time of packets being queued if the queue is full and there are video packets, the oldest video packet is discarded, otherwise the oldest audio packet is discarded. At the time of transmission the algorithm for SBPN2 is followed. This means that packets aren't always transmitted if packets are expired, unlike ring buffers which transmits a packet even if the packet has expired.

Figure 7.8 and Figure 7.9 shows that SBPN2 with ring buffer improves quality when compared to SBPN2 – both using a queue length of 5. There is a large improvement across most of the range of bit rates on the constrained link. Table 7.3 shows the breakdown of the impact of loss and delay. This table shows that the delay results for both versions are very similar but the loss rate is much lower for SBPN2 with ring buffers and that this gives the improvement in quality. SBPN2 with ring buffers still discards packets if they have expired at time of transmission unlike ring buffers without SBPN.

For this improvement in loss rate to occur it means that the packets in the queue must be more recent if they are not being discarded. Ring buffers

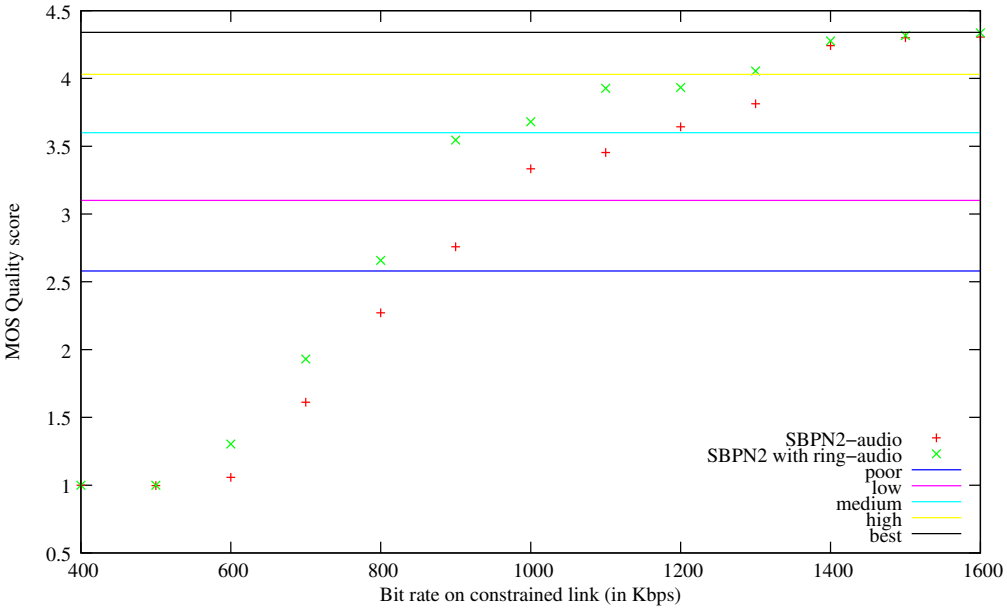


Figure 7.9: SBPN2 vs SBPN2 with ring - 80 ms RTT, queue length 5

Table 7.3: Loss and delay with queue length 5, 80 ms RTT

	SBPN2					SBPN2 with ring				
Rate	Loss	Delay	I_e	I_d	MOS	Loss	Delay	I_e	I_d	MOS
100	97.69%	2016	82.52	250.70	1.00	97.89%	2000	82.58	248.51	1.00
200	93.78%	943	81.37	106.88	1.00	92.50%	964	80.99	109.72	1.00
300	72.24%	619	74.14	63.45	1.00	76.35%	622	75.66	63.83	1.00
400	63.45%	468	70.59	43.26	1.00	67.53%	472	72.29	43.74	1.00
500	48.46%	373	63.38	30.41	1.00	48.67%	376	63.49	30.91	1.00
600	44.09%	308	60.90	21.73	1.06	30.20%	304	51.30	21.21	1.30
700	26.28%	266	47.94	16.19	1.61	19.17%	267	40.64	16.32	1.93
800	17.18%	234	38.24	11.82	2.27	11.95%	234	30.81	11.82	2.66
900	13.22%	205	32.79	7.93	2.76	5.24%	204	17.41	7.89	3.55
1000	8.43%	184	24.52	5.09	3.33	5.23%	183	17.37	5.01	3.68
1100	7.77%	167	23.19	4.00	3.45	3.51%	166	12.68	3.99	3.93
1200	6.12%	152	19.53	3.66	3.64	3.56%	154	12.84	3.68	3.93
1300	4.70%	142	16.00	3.41	3.81	2.62%	142	9.94	3.41	4.06
1400	1.12%	129	4.66	3.10	4.24	0.82%	129	3.47	3.10	4.28
1500	0.68%	117	2.90	2.81	4.30	0.53%	117	2.28	2.82	4.32
1600	0.67%	110	2.88	2.65	4.31	0.38%	110	1.68	2.64	4.34
1700	0.61%	102	2.64	2.45	4.32	0.47%	102	2.05	2.46	4.33
1800	0.81%	95	3.45	2.29	4.30	0.46%	95	2.01	2.29	4.34
1900	0.42%	89	1.85	2.14	4.34	0.40%	89	1.77	2.15	4.35
2000	0.42%	84	1.81	2.02	4.35	0.36%	84	1.57	2.02	4.35

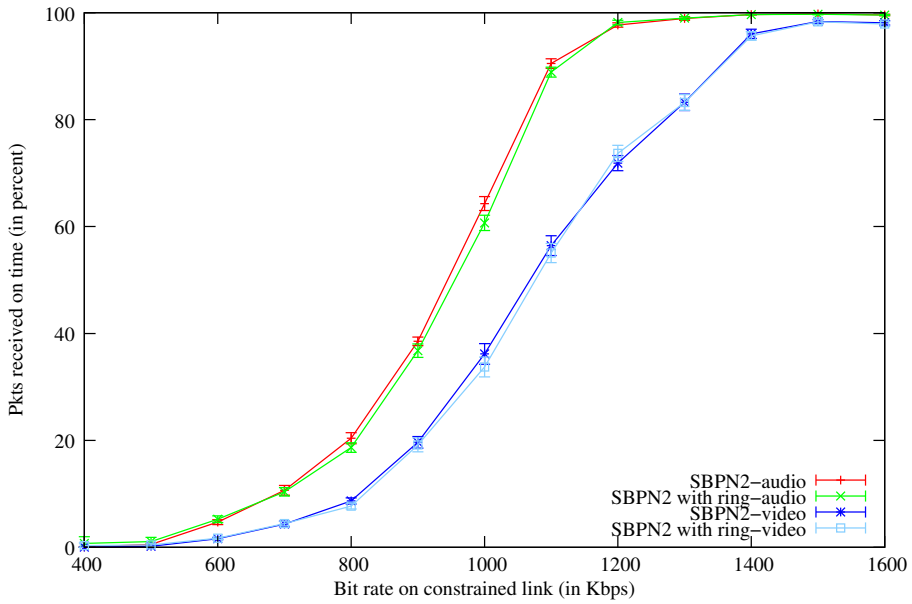


Figure 7.10: SBPN2 vs SBPN2 with ring - 80 ms RTT, queue length 32

always adds a new packet to a queue at the time of queueing, rather than discarding the packet if the queue is full. This behaviour gives the improvements demonstrated and meets the needs of real time media applications which need timely data.

Figure 7.10 and Figure 7.11 shows the equivalent results when a queue length of 32 is used. SBPN2 has slightly higher on-time arrival, but there is not a clear difference for quality.

The difference for on-time arrival may be due to SBPN2 with ring buffers discarding video packets at time of insertion in preference to audio. These video packets could be more recent than the audio packets in the queue. This would lead to audio packets being discarded due to expiry, which would otherwise not have occurred and thus reducing the send rate.

Tests were also carried out for 30 milliseconds RTT and 150 milliseconds RTT with a queue length of 5 and a queue length of 32. These showed similar results to the tests conducted with 80 milliseconds RTT.

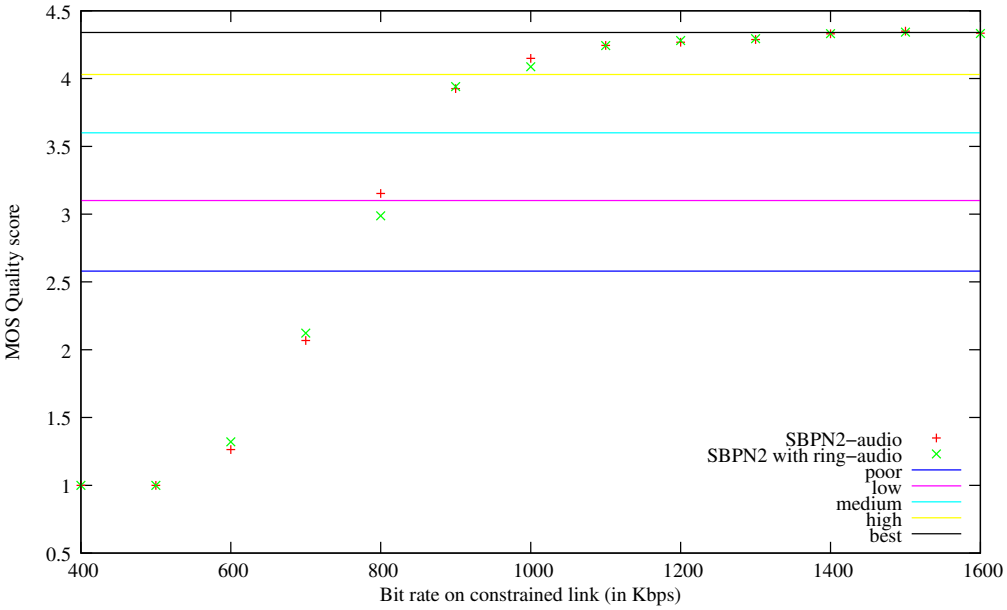


Figure 7.11: SBPN2 vs SBPN2 with ring - 80 ms RTT, queue length 32

7.5 Comparing Ring Buffers to SBPN2 with Ring Buffers

In the previous section (7.4) SBPN2 was compared to SBPN2 with ring buffers. Figure 7.12 and Figure 7.13 show ring buffers compared to SBPN2 with ring buffers for a queue length of 5. These graphs show similar results for the two algorithms and show the benefit of the ring buffer inserting recent data into the queue so that the transmit rate is not reduced.

Figure 7.14 and Figure 7.15 compare the two algorithms with a queue length of 32. These results show the same trend as occurred with a queue length of 5.

The tests were then run with an RTT of 30 milliseconds and 150 milliseconds. The tests with an RTT of 30 milliseconds showed no discernible difference between the algorithms. Figure 7.16 and Figure 7.17 show the results for 150 millisecond RTT and show that there is a small improvement for SBPN2 with ring buffers when compared to ring buffers.

The results shown in Figure 7.17 that for longer RTTs the advantage is increased for SBPN2 with ring buffers as the algorithm will discard more packets

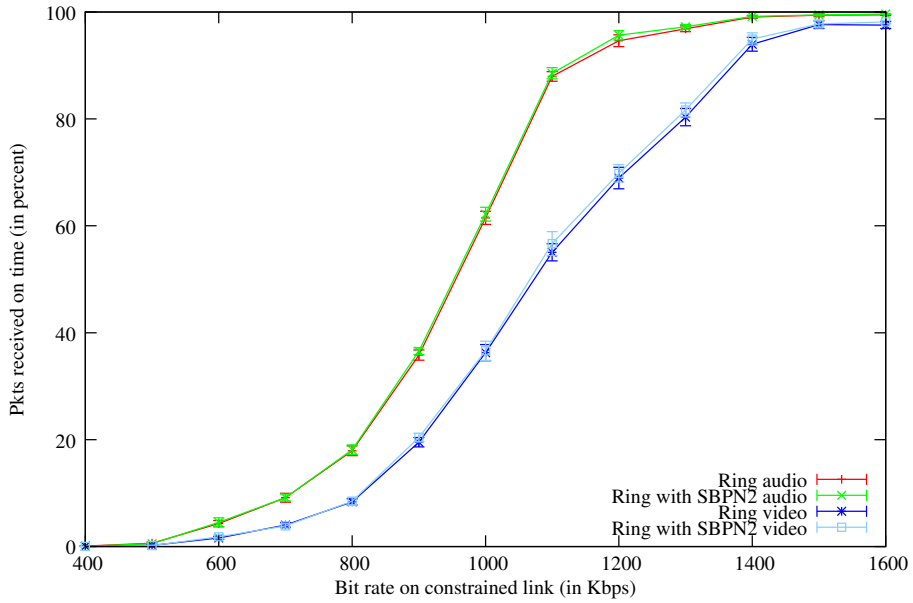


Figure 7.12: Ring buffers vs SBPN2 with ring - 80 ms RTT, queue length 5

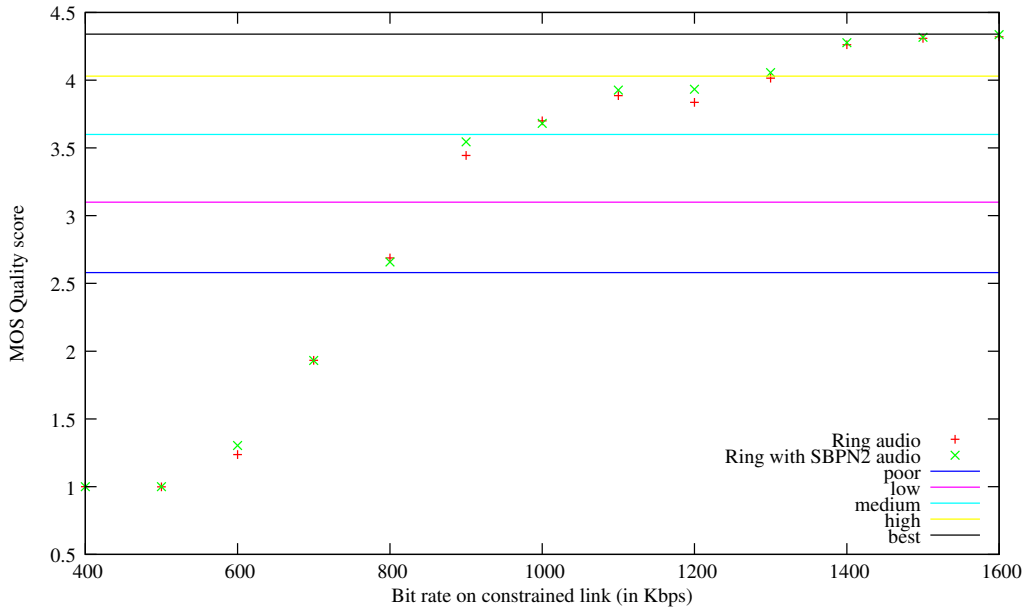


Figure 7.13: Ring buffers vs SBPN2 with ring - 80 ms RTT, queue length 5

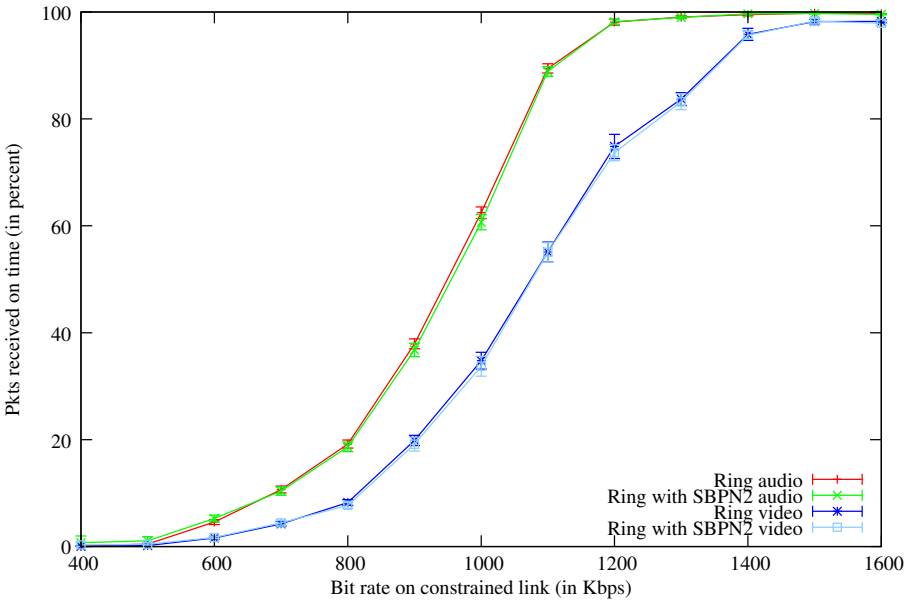


Figure 7.14: Ring buffers vs SBPN2 with ring - 80 ms RTT, queue length 32

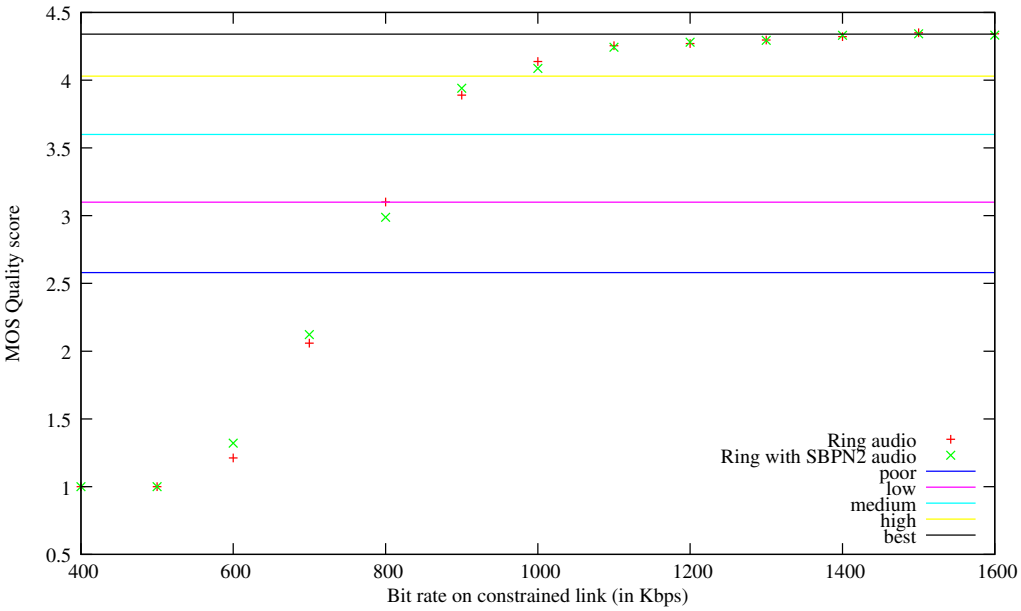


Figure 7.15: Ring buffers vs SBPN2 with ring - 80 ms RTT, queue length 32

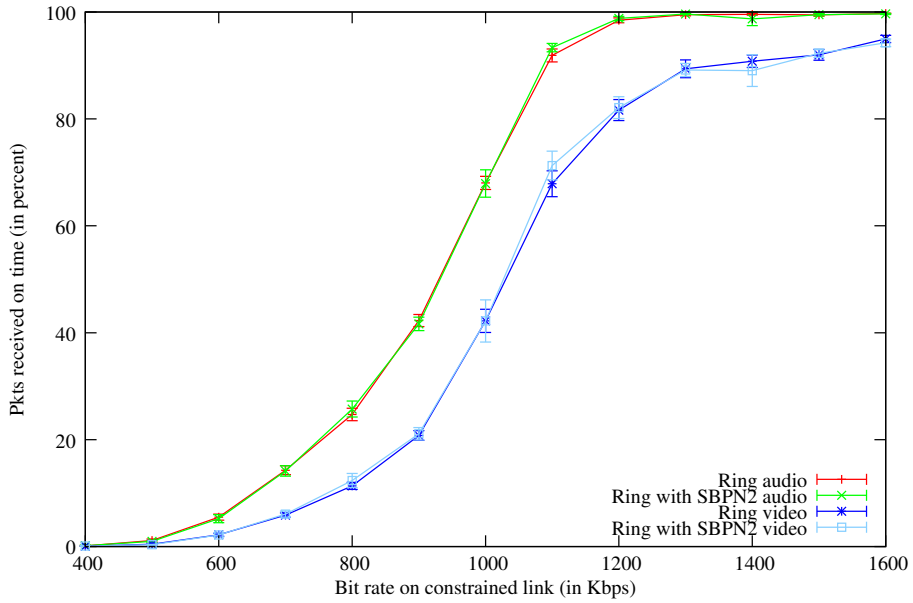


Figure 7.16: Ring buffers vs SBPN2 with ring - 150 ms RTT, queue length 32

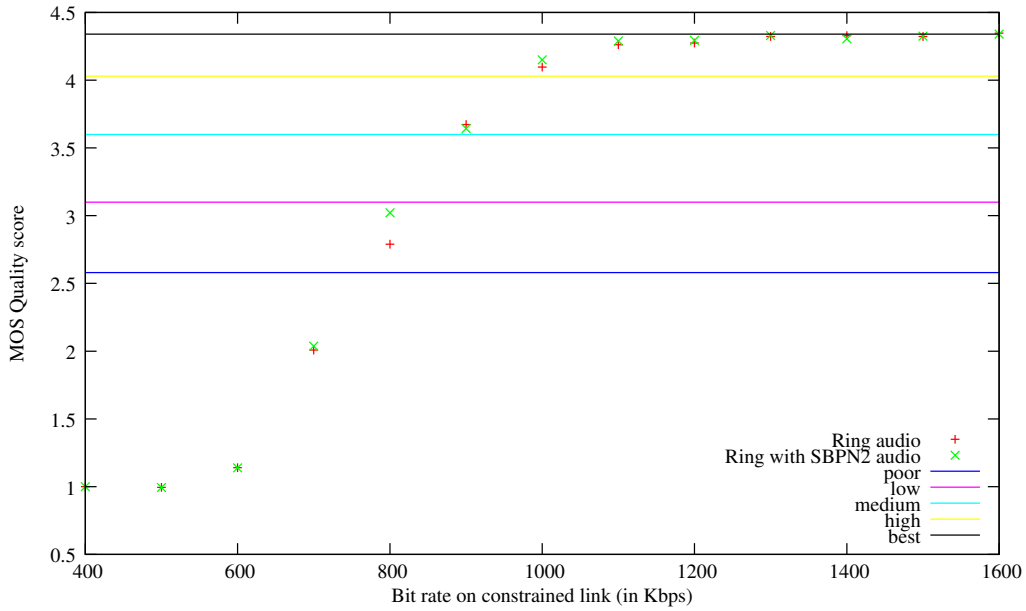


Figure 7.17: Ring buffers vs SBPN2 with ring - 150 ms RTT, queue length 32

Table 7.4: Loss and delay with queue length 32, 150 ms RTT

	Ring buffers					SBPN2 with ring				
Rate	Loss	Delay	I_e	I_d	MOS	Loss	Delay	I_e	I_d	MOS
100	97.18%	1955	82.38	242.47	1.00	97.00%	1948	82.32	241.55	1.00
200	92.68%	933	81.05	105.52	1.00	92.32%	926	80.94	104.61	1.00
300	82.84%	610	77.92	62.24	1.00	82.64%	620	77.85	63.54	1.00
400	62.33%	459	70.11	41.96	1.00	64.82%	458	71.17	41.83	1.00
500	49.32%	366	63.84	29.54	1.00	48.01%	369	63.13	29.90	0.99
600	39.79%	296	58.24	20.22	1.14	38.95%	301	57.69	20.84	1.14
700	19.32%	254	40.81	14.53	2.01	18.63%	256	40.01	14.74	2.04
800	11.16%	225	29.51	10.61	2.79	8.85%	223	25.34	10.38	3.02
900	4.44%	200	15.30	7.27	3.67	4.64%	200	15.86	7.36	3.64
1000	2.01%	178	7.91	4.31	4.10	1.61%	175	6.48	4.20	4.15
1100	0.77%	160	3.26	3.85	4.26	0.54%	156	2.36	3.75	4.29
1200	0.74%	145	3.16	3.48	4.27	0.57%	144	2.47	3.47	4.29
1300	0.38%	133	1.66	3.18	4.32	0.31%	134	1.37	3.21	4.33
1400	0.38%	124	1.65	2.98	4.33	0.59%	126	2.53	3.02	4.30
1500	0.47%	117	2.03	2.80	4.32	0.45%	116	1.94	2.79	4.33
1600	0.27%	110	1.19	2.64	4.35	0.34%	111	1.48	2.68	4.34
1700	0.27%	104	1.21	2.49	4.35	0.28%	105	1.24	2.52	4.35
1800	0.37%	101	1.60	2.42	4.34	0.22%	98	0.99	2.36	4.36
1900	0.20%	93	0.90	2.24	4.36	0.38%	95	1.66	2.27	4.35
2000	0.24%	90	1.07	2.16	4.36	0.23%	91	1.03	2.17	4.36

Table 7.5: Loss and delay with queue length 32, 80 ms RTT

	Ring buffers					SBPN2 with ring				
Rate	Loss	Delay	I_e	I_d	MOS	Loss	Delay	I_e	I_d	MOS
100	97.38%	1971	82.43	244.62	1.00	97.91%	1997	82.58	248.12	1.00
200	93.18%	925	81.19	104.45	1.00	92.56%	965	81.01	109.91	1.00
300	76.60%	631	75.75	65.09	1.00	77.09%	618	75.92	63.25	1.00
400	58.44%	469	68.37	43.32	1.00	61.28%	457	69.65	41.68	1.00
500	50.71%	376	64.58	30.84	1.00	51.00%	369	64.72	29.93	1.00
600	33.90%	305	54.18	21.42	1.21	29.50%	304	50.73	21.24	1.32
700	17.14%	266	38.19	16.10	2.06	16.10%	266	36.85	16.16	2.12
800	7.33%	235	22.25	11.93	3.10	8.32%	236	24.30	12.06	2.99
900	2.53%	205	9.64	7.95	3.89	2.09%	206	8.17	8.16	3.94
1000	1.42%	185	5.79	5.24	4.14	1.80%	185	7.18	5.30	4.09
1100	0.79%	167	3.37	4.00	4.25	0.88%	167	3.73	4.01	4.24
1200	0.76%	148	3.24	3.55	4.27	0.64%	153	2.76	3.67	4.28
1300	0.56%	142	2.44	3.40	4.30	0.59%	142	2.53	3.41	4.29
1400	0.42%	129	1.82	3.10	4.32	0.33%	128	1.43	3.07	4.33
1500	0.21%	118	0.94	2.82	4.35	0.28%	118	1.22	2.83	4.34
1600	0.34%	110	1.43	2.65	4.34	0.41%	111	1.79	2.66	4.33
1700	0.39%	102	1.72	2.45	4.34	0.29%	102	1.27	2.45	4.35
1800	0.30%	96	1.31	2.29	4.35	0.30%	95	1.33	2.29	4.35
1900	0.31%	90	1.38	2.16	4.35	0.33%	89	1.44	2.14	4.35
2000	0.27%	85	1.20	2.04	4.36	0.32%	84	1.39	2.01	4.36

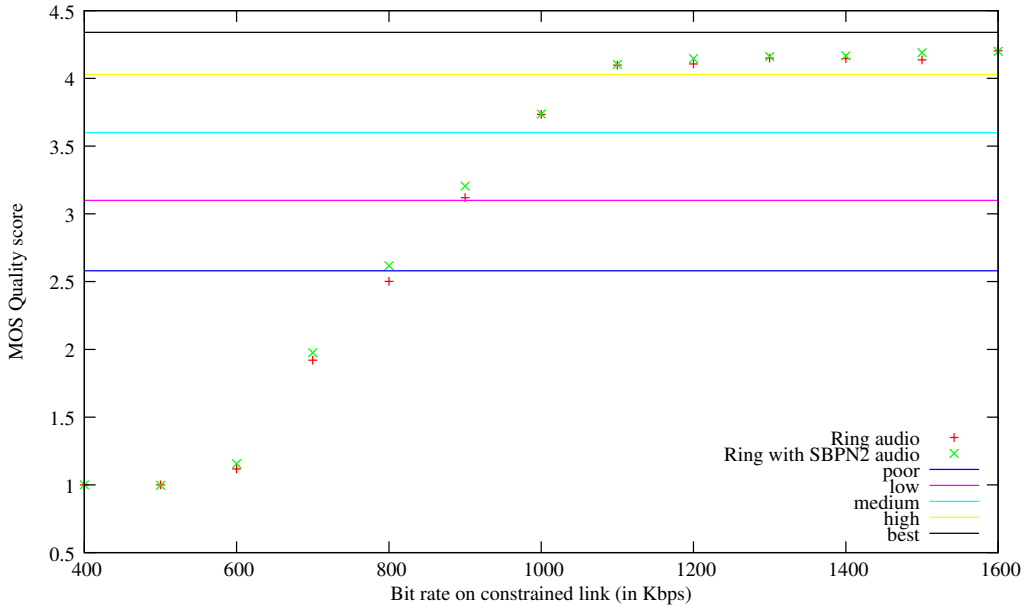


Figure 7.18: Ring buffers vs SBPN2 with ring - 150 ms RTT, queue length 5

that are due to expire at longer RTTs, where the pure ring buffer implementation does not consider packet expiry time. This can be seen in Table 7.4 where the data shows that in the range of 800-1100 Kbps overall quality score is higher and loss rate is lower with delay around the same. This is in contrast to Table 7.5 where the quality and loss are similar for both algorithms as packet expiry is less of an issue with a lower RTT.

The tests were repeated with a queue length of 5 and the quality results are shown in Figure 7.18 which also shows improvements being made with a longer RTT.

The results obtained in this section correspond with the results in the previous section that SBPN2 with ring buffers can have lower loss than ring buffers under the right network conditions.

7.6 Summary

In this chapter elements of Kohler and Lai's research examining the use of ring buffers were combined with the underlying implementation for SBPN. Kohler and Lai worked on a queue length of 3 packets and this was extended to longer

queue lengths and used the same two priorities (audio and video) as our earlier research.

It was shown that adapting the use of ring buffers gave substantially better results than SBPN2 for queue sizes of 5, but not for queue sizes of 32. This is illustrated in Figure 7.3 compared to Figure 7.5. The reason for this is that ring buffers replace the oldest packet when a queue is full, as opposed to discarding the new packet. With SBPN2 discarding expired packets CCID3 slows down the allowable transmit rate, due to lower actual transmit rate.

In Section 7.4 SBPN2 and ring buffers are combined. This gives improvements for the queue size of 5, even though the algorithm still discards packets at the time of transmission. This shows that the improvements made by examining packets at the time of queueing outweigh SBPN2 discarding packets and the rate being reduced. The results for queue length 32 do not differ significantly between the algorithms as the increased queue length allows SBPN2 to work effectively without being affected by transmit rate reduction.

When SBPN2 with ring buffers was compared to ring buffers on their own it gave better results for long RTTs as shown in Figure 7.17 and Figure 7.18. This is because the ring buffer algorithm does not consider how useful the packet will be to the receiver, apart from audio being more important than video. With a long RTT SBPN2 will discard more packets that would not make it to the receiver in time to be used and can thus send more relevant data instead.

It can be seen from this chapter that ring buffers builds upon SBPN2 significantly for smaller queue sizes and SBPN2 works well with larger queue sizes and has advantages for longer RTTs.

In the following chapter the overall contribution of SBPN is summarised and avenues for future work are explored.

Chapter 8

Conclusion

8.1 Introduction

This chapter reviews the overall potential of the Send the Best Packet Next (SBPN) algorithm and also gives future possible direction and outlines the benefits given through SBPN.

The practical work carried out in implementing and testing SBPN are summarised in Section 8.2.

The best results obtained from SBPN are given and demonstrated to be substantially better than unmodified DCCP. These results are shown in Section 8.3.

In Section 8.4 a number of ideas for possible further work and future research are outlined.

Section 8.5 summarises the contribution that SBPN makes and how it improves the performance of real time media.

8.2 Practical work

The results from testing SBPN were based on using machines on a network, rather than via a simulator. As part of this DCCP was implemented in the official Linux kernel and released, in conjunction with other implementers, so that it now forms part of the source code for the standard Linux kernel. This

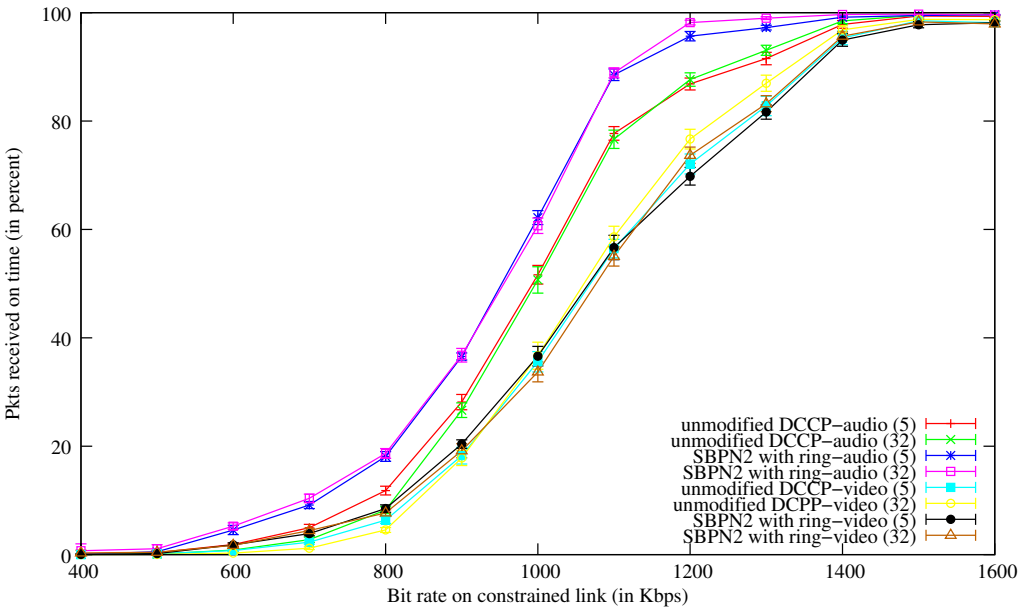


Figure 8.1: Unmodified DCCP vs SBPN2 with ring - 80 ms RTT, queue lengths 5 and 32

implementation was tested on both local networks and the open Internet.

SBPN was then produced as an extension to CCID3 in the Linux kernel.

A media model, based on applications in use, was created to allow traffic to be generated that was similar to actual use of real time media applications. A test setup was created that allowed network conditions to vary including delay, loss and congestion and modifications to open source tools in this area were given back to the community and incorporated in upstream releases. Measurement tools were created to analyse the results and to measure time differences between machines to higher resolution than Network Time Protocol (NTP).

Feedback from implementing DCCP and CCID3 was given to the IETF and used to help improve the specifications for these standards.

8.3 Best results

The overall improvement made by SBPN is shown in Figure 8.1 and Figure 8.2 which are a combination of the best results from earlier chapters. With unmodified DCCP queue length 5 does worse than queue length 32 on quality.

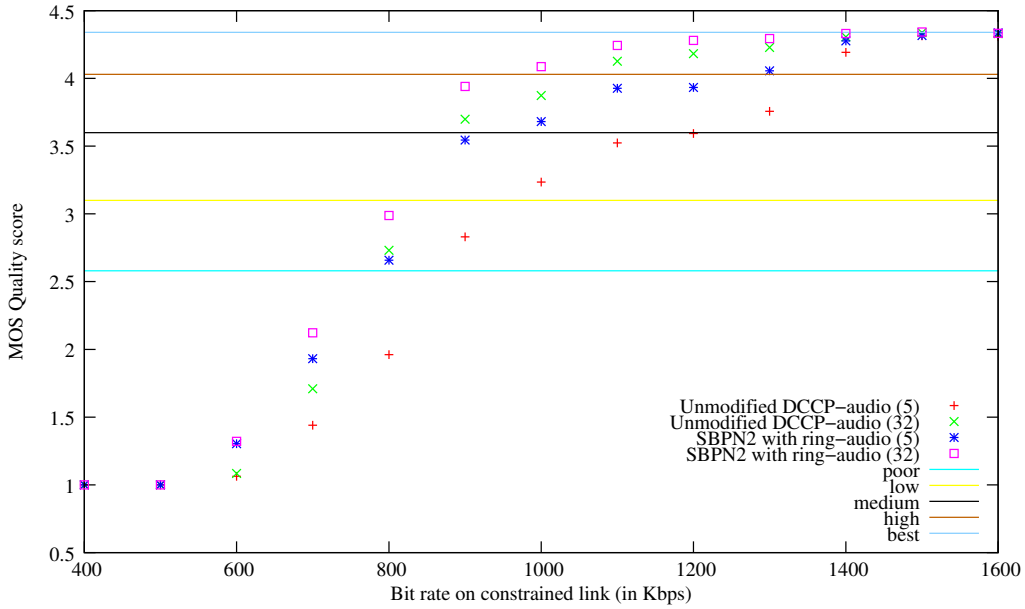


Figure 8.2: Unmodified DCCP vs SBPN2 with ring - 80 ms RTT, queue length 32

This was discussed in Section 5.3.2 which showed that DCCP slowed down the allowable transmit rate with shorter queues. It is interesting to note that ontime arrival does not differ between queue lengths of 5 and 32, but quality does alter significantly.

The quality is improved significantly by SBPN2 with ring buffers. The biggest improvement occurs with queue length 5, while the best overall quality occurs with queue length 32 and SBPN2 with ring buffers. Smaller queue lengths are affected by TFRC slowing down the transmission rate based on there being empty buffers at times as packets have been discarded as expired, and this is discussed further in the following section as an area of future research.

The combination of SBPN2 and ring buffers works well as both algorithms improve quality but via different methods. SBPN2 makes decisions on which packet to send at the time of transmission and to see whether they will expire in transit, and ring buffers look at which packets should be in the queue when the application passes them to the network protocol. To put it another way ring buffers makes decisions on packets as they go into the queue, and SBPN2

makes decisions on packets as they leave the queue.

8.4 Further work and future research

There are a number of areas that can be investigated to see how SBPN can be improved or works under different conditions and these are detailed in this section. From a theoretical approach the most improvement could be made by looking to sending all packets as described in Section 8.4.3. From a practical approach the most gains could be made by modifying applications as described in Section 8.4.2.

8.4.1 Changing bandwidth

The work in this thesis was based upon fixed network bandwidth. Further work could be carried out with variable network bandwidth with some example scenarios being:

- slow deterioration or improvement of available bandwidth
- sudden deterioration or improvement of available bandwidth with two scenarios — brief and permanent
- random fluctuation in available bandwidth

Carrying out work in this area would show whether SBPN can help in changeable network situations. In many cases codecs will change the bit rate used, and SBPN may be able to help before the codec is able to change the bit rate.

8.4.2 Modifying applications

The next area of future research suggested is converting an application from using UDP and/or TCP with conventional congestion control methods to using DCCP with SBPN communicating network conditions to the application.

Part of this further research would also look at how the protocol stack would notify rate changes through the API to the application and how the codecs would then adapt to this change. This would also involve how the application would react to varying network bandwidth rates e.g. should the application drop resolution, drop frame rate, decrease frame quality or stop altogether if the desired rates cannot be obtained.

A key part of this would be defining an API that works in a similar manner to existing APIs for TCP and UDP, or for higher level protocols such as RTP for ease of implementation for application developers. Such an API would be two way — communicating changes both to and from the network stack and the media application. The application would specify parameters such as minimum and maximum bandwidth, loss, jitter and any changes to these through the API. The network stack would provide to the application data such as available bandwidth, RTT, loss events and loss rates.

In effect the application, transport and network layers would be coupled together much more tightly to see how much further improvement could be made.

The work in this thesis has been carried out with DCCP and since the research was started, DCCP has not become prevalent on the Internet. This is probably due to the lack of DCCP support in commercial operating systems and the difficulty of DCCP to pass through firewalls. DCCP over UDP [93] offers a possible solution to this and will enable the use of DCCP by encapsulating the packets and transmitting them over UDP. SBPN could just as easily use DCCP over UDP instead of DCCP.

8.4.3 Sending all packets

A consistent finding through this thesis was that the allowable transmit rate drops if SBPN discards expired packets, with a subsequent impact on quality. It would be worth creating a variant of SBPN which never discards packets, and also a variant which sends “keep alive” packets. In Sathiaseelan [101]

the authors find that a newer version of TFRC specified in RFC 5348 [48] resolved the need to send keep alive packets for their media flows, so it would be worthwhile to use this version in our tests also.

It would also be worthwhile to look at implementing newer versions of TCP congestion control such as BIC [111] or Cubic [46] as CCIDs for DCCP and testing whether this would improve the performance of SBPN by alleviating the issue of allowable transmit rate being reduced.

8.4.4 Testing with multiple layers

The testing in this thesis was carried out with two layers — audio and video. The implementation of SBPN allowed further layers but this was not tested. Codecs often separate video into multiple layers and SBPN could be tested with this.

A form of weighting could also be considered for multiple layer queues. This could calculate a score based on packet type, expiry time and any other variables which would then be used as a basis for sorting the transmit queue.

8.4.5 Subjective measurement

This thesis used objective measures by using measures such as MOS and R-score. It would be worth validating these measures by also testing subjectively with people rating the quality. This could be done by modifying an open source application, such as Ekiga, to use SBPN and then testing this with participants rating with and without SBPN in double blind tests to eliminate bias.

8.4.6 Other areas of research

Testing could be carried out with other modern codecs such as G.729 [60] and Celt [108] to see if SBPN gives similar improvements with these codecs also. In theory SBPN is codec independent as shown in Section 4.6.2, but it would be useful to test this assumption.

It would also be useful to see if DCCP and SBPN could differentiate between loss occurring as a result of congestion, and loss occurring due to corruption (such as on a wireless link). For TCP there has been variants such as Vegas [23], Westwood [73], Veno [42] and Santa Cruz [88] which aim to improve the situation for lossy links such as wireless. These algorithms could potentially be added to DCCP as CCIDs and then tested to see how they interact with SBPN.

SBPN could be tested in conjunction with QoS measures such as diffserv to see if combining the two approaches gives further possible gains in quality.

8.5 Summary of Contributions

A new class of algorithms have been introduced that determines packet transmission on a "send the best packet next" principle, rather than on a First In First Out (FIFO) basis that is normally used for network transmission. These algorithms apply information supplied by applications within the network system.

Our unique contribution is that we combine priority queues with tracking of expiry times which take into account the Round Trip Time (RTT). This allows selective protection to part of a media stream, audio, and is done in a manner that does not greatly impact the application developer. This is tested and implemented using the DCCP transport protocol [66] which is congestion controlled but unreliable. Ideas are taken from the ring buffer implementations of Kohler and Lai [69] and used to further improve the algorithms, particularly for smaller queue lengths.

A number of other ideas were also tested such as TFRC Faster Restart and LIFO queues. These did not give a significant improvement and were not investigated further.

The tests that were carried out were based on practical work implemented using the Linux kernel and a number of other open source projects and given

back to the open source projects. A model of media applications was built and also a testing framework.

This thesis shows that it is possible to gain a substantial improvement to real time media applications when congestion is experienced by using the SBPN algorithm. This is demonstrated by showing improvement in the number of audio packets "received on time" when tested with competing TCP flows, which will give a higher quality real time media experience. These results have been shown across a range of RTTs and with differing queue lengths. Using the best algorithm (SBPN2 combined with ring buffers) shows "audio on time" improves by over 10% across most of the range while video on time does not decrease or decreases by a much smaller amount as shown in Figure 8.1. Audio quality as measured by R score has improved substantially as shown in Figure 8.2 — as an example at 1000 Kbps and a queue length of 5 quality was improved from medium to high.

As real improvement has been demonstrated the recommendation of this thesis is to replace UDP as transport in media applications with DCCP and SBPN as shown in Section 5.3. As the algorithms proposed in this thesis are made at the sender end only, servers can implement these algorithms and give a significant benefit to clients without a change being required by the clients. As the changes are server side only the SBPN algorithm can be widely deployed over a range of services such as real time IP Television (IPTV) and video conferencing.

Alternatively SBPN could also be applied over more widely accepted protocols such as UDP if effective congestion control and RTT tracking were used with UDP, and the same benefits would potentially be obtained.

This thesis has shown that by the application supplying information to the transport layer, significantly improvements can be made to audio quality and thus overall user experience.

Appendix A

Problem statement for DCCP

Network Working Group
Request for Comments: 4336
Category: Informational

S. Floyd
ICIR
M. Handley
UCL
E. Kohler
UCLA

March 2006

Problem Statement for the
Datagram Congestion Control Protocol (DCCP)

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

This document describes for the historical record the motivation behind the Datagram Congestion Control Protocol (DCCP), an unreliable transport protocol incorporating end-to-end congestion control. DCCP implements a congestion-controlled, unreliable flow of datagrams for use by applications such as streaming media or on-line games.

Table of Contents

1. Introduction 2
2. Problem Space 3
2.1. Congestion Control for Unreliable Transfer 4
2.2. Overhead 6
2.3. Firewall Traversal 6
2.4. Parameter Negotiation 7
3. Solution Space for Congestion Control of Unreliable Flows 7
3.1. Providing Congestion Control Above UDP 8
3.1.1. The Burden on the Application Designer 8
3.1.2. Difficulties with ECN 8
3.1.3. The Evasion of Congestion Control 10
3.2. Providing Congestion Control Below UDP 10
3.2.1. Case 1: Congestion Feedback at the Application 11
3.2.2. Case 2: Congestion Feedback at a Layer Below UDP ...11
3.3. Providing Congestion Control at the Transport Layer 12
3.3.1. Modifying TCP? 12
3.3.2. Unreliable Variants of SCTP? 13
3.3.3. Modifying RTP? 14
3.3.4. Designing a New Transport Protocol 14
4. Selling Congestion Control to Reluctant Applications 15
5. Additional Design Considerations 15
6. Transport Requirements of Request/Response Applications 16
7. Summary of Recommendations 17
8. Security Considerations 18
9. Acknowledgements 18
Informative References 19

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 streaming audio, Internet telephony, and multiplayer and massively multiplayer on-line games, 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 [RFC2914, RFC3714].

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 will 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 describe why a new protocol is the best solution for the more general problem of bringing congestion control to unreliable flows of unicast datagrams, and discuss briefly subsidiary requirements for mobility, defense against Denial of Service (DoS) attacks and spoofing, interoperation with RTP, and interactions with Network Address Translators (NATs) and firewalls.

One of the design preferences that we bring to this question is a preference for a clean, understandable, low-overhead, and minimal protocol. As described later in this document, this results in the design decision to leave functionality such as reliability or Forward Error Correction (FEC) to be layered on top, rather than provided in the transport protocol itself.

This document began in 2002 as a formalization of the goals of DCCP, the Datagram Congestion Control Protocol [RFC4340]. We intended DCCP to satisfy this problem statement, and thus the original reasoning behind many of DCCP's design choices can be found here. However, we believed, and continue to believe, that the problem should be solved whether or not DCCP is the chosen solution.

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 following:

- The potential for non-congestion-controlled datagram flows to cause congestion collapse of the network. (See Section 5 of [RFC2914] and Section 2 of [RFC3714].)

- The difficulty of correctly implementing effective congestion control mechanisms for unreliable datagram flows.

- The lack of a standard solution for reliably transmitting congestion feedback for an unreliable data flow.

- The lack of a standard solution for negotiating Explicit Congestion Notification (ECN) [RFC3168] usage for unreliable flows.

- The lack of a choice of TCP-friendly congestion control mechanisms.

We assume that most application writers would use congestion control for long-lived unreliable flows if it were available in a standard, easy-to-use form.

More minor issues include the following:

- 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

We aim to bring easy-to-use congestion control mechanisms to applications that generate large or long-lived flows of unreliable datagrams, such as RealAudio, Internet telephony, and multiplayer games. Our motivation is to avoid congestion collapse. (The short flows generated by request-response applications, such as DNS and SNMP, 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, among them:

- Startup Delay: they wish to avoid the delay of a three-way handshake before initiating data transfer.

- 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 issues, applications that generate large or long-lived flows of datagrams, such as media transfer and games, mostly care about controlling the trade-off 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 an interactive game or virtual-reality 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, some control is given up 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, TCP-Friendly Rate Control (TFRC) [RFC3448] rather than strict TCP-like congestion control. (TFRC has been standardized in the IETF as a congestion control mechanism that adjusts its sending rate more smoothly than TCP does, while maintaining long-term fair bandwidth sharing with TCP [RFC3448].)

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 5.6 Kbps data stream but wanting moderately low delay may send a packet every 20 ms, sending only 14 data bytes in each packet. In addition, 20 bytes is taken up by the IP header, with additional bytes for transport and/or application headers. Clearly, it is desirable for such an application to have a low-overhead transport protocol header.

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 the Real Time Streaming Protocol (RTSP) [RFC2326], SIP [RFC3261], 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 after being redirected or relayed to other machines.

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. Although, for bi-directional flows, the firewall can open a bi-directional hole if it receives a UDP packet from inside the firewall, 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 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 for congestion control, which may map into different congestion feedback. Examples include ECN capability and desired congestion control dynamics (the choice of congestion control algorithm and, therefore, 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 in 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. This 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 [RFC2525, PF01] 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

There is a second problem with providing congestion control above UDP: 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 be useful even in the absence of network support. 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 ECN contract is that senders who set the ECT codepoint must respond to Congestion Experienced (CE) codepoints by reducing their sending rates. Therefore, the ECT codepoint can only safely be set in the packet header of a UDP packet if the following is guaranteed:

- if the CE codepoint is set by a router, the receiving IP layer will pass the CE status to the UDP layer, which will pass it 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.

In the absence of more fine-grained mechanisms for dealing with a period of sustained congestion, one possibility would be for routers to discontinue using ECN with UDP packets during the congested period, and to use ECN only with TCP or DCCP packets. This would be a reasonable response, for example, if TCP or DCCP flows were found

to be more likely to be using conformant end-to-end congestion control than were UDP flows. If routers were to adopt such a policy, then DCCP flows could be more likely to receive the benefits of ECN in times of congestion than would UDP flows.

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, with applications using 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 (CM) [RFC3124].

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 [BRS99] 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. Thus, this design limits the choice of congestion control mechanisms available to applications while simultaneously burdening the applications with implementations of congestion feedback.

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 8-byte header for UDP itself. Unless this header were an IP option (which is likely to cause problems for many IPv4 routers), 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, and the modified protocol would face deployment challenges similar to those of an entirely new protocol.

To use congestion feedback at a layer below UDP most effectively, the semantics of the UDP socket Application Programming Interface (API) would also need changing, both to support a late decision on what to send and to provide access to sequence numbers (so that the application wouldn't need to duplicate them for its own purposes). Thus, the socket API would no longer look like UDP to 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 the Stream Control Transmission Protocol (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 might be to create an unreliable variant of TCP, with reliability layered on top for applications desiring reliable delivery. However, our requirement is not simply for TCP minus 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, not datagrams. This would complicate both congestion feedback and any attempt to allow the application to decide, at transmission time, what information should go into a packet. 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.

It seems best simply to leave TCP as it is, and to create a new congestion control protocol for unreliable transfer. This is especially true since any change to TCP, no matter how small, takes an inordinate amount of time to standardize and deploy, given TCP's importance in the current Internet and the historical difficulty of getting TCP implementations right.

3.3.2. Unreliable Variants of SCTP?

SCTP, the Stream Control Transmission Protocol [RFC2960], 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, of 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 (ECNE) and Congestion Window Reduced (CWR) 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.

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.

Finally, the SCTP Partial Reliability extension [RFC3758] allows a sender to selectively abandon outstanding messages, which ceases retransmissions and allows the receiver to deliver any queued messages on the affected streams. This service model, although well-suited for some applications, differs from, and provides the application somewhat less flexibility than, UDP's fully unreliable service.

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. We would argue against this. SCTP is well-suited for applications that desire limited retransmission with multistream or multihoming support. Adding support for fully-unreliable variants, multiple congestion control profiles, and reduced single-stream headers would risk introducing unforeseen interactions and make further

modifications ever more difficult. We have chosen instead to implement a minimal protocol, designed for fully-unreliable datagram service, that provides only end-to-end congestion control and any other mechanisms that cannot be provided in a higher layer.

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?

When RTP lives above UDP, modifying it to support congestion control might create 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, such as a reduced RTP header that removes any fields redundant with the new protocol's headers.

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 as 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.

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 design requirements include a choice between different forms of congestion control, moderate overhead in the size of the packet header, and the use of Explicit Congestion Notification (ECN) and the ECN nonce [RFC3540], which provide positive benefit to the application itself.

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 [RFC2309, 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 toward 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 multihoming 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 multihoming and mobility into the protocol itself. The current version of DCCP [RFC4340] includes no multihoming or mobility support.

- 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 offer additional protection against DoS attacks. Thus, in designing 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 any necessary or desirable changes in RTP when it is used over the new protocol, such as reduced RTP headers.

- 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 time, and providing applications with some information about packet sequence numbers.

- Interactions with NATs and firewalls: NATs and firewalls don't interact well with UDP, with its lack of explicit flow setup and teardown and, in practice, the lack of well-known ports for many UDP applications. Some of these issues are application specific; others should be addressed by the transport protocol itself.

- 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 this document addresses the requirements relevant to our problem area that were brought up at the RUTS BOF.

6. Transport Requirements of Request/Response Applications

Up until now, this document has discussed the transport and congestion control requirements of applications that generate long-lived, large flows of unreliable datagrams. This section discusses briefly the transport needs of another class of applications, those of request/response transfers where the response might be a small number of packets, with preferences that include both reliable delivery and a minimum of state maintained at the ends. The reliable delivery could be accomplished, for example, by having the receiver re-query when one or more of the packets in the response is lost. This is a class of applications whose needs are not well-met by either UDP or by TCP.

Although there is a legitimate need for a transport protocol for such short-lived reliable flows of such request/response applications, we believe that the overlap with the requirements of DCCP is almost non-existent and that DCCP should not be designed to meet the needs of these request/response applications. Areas of non-compatible requirements include the following:

- Reliability: DCCP applications don't need reliability (and long-lived applications that do require reliability are well-suited to TCP or SCTP). In contrast, these short-lived request/response applications do require reliability (possibly client-driven reliability in the form of requesting missing segments of a response).

- Connection setup and teardown: Because DCCP is aimed at flows whose duration is often unknown in advance, it addresses interactions with NATs and firewalls by having explicit handshakes for setup and teardown. In contrast, the short-lived request/response applications know the transfer length in advance, but cannot tolerate the additional delay of a handshake for flow setup. Thus, mechanisms for interacting with NATs and firewalls are likely to be completely different for the two sets of applications.

- Congestion control mechanisms: The styles of congestion control mechanisms and negotiations of congestion control features are heavily dependent on the flow duration. In addition, the preference of the request/response applications for a stateless server strongly impacts the congestion control choices. Thus, DCCP and the short-lived request/response applications have rather different requirements both for congestion control mechanisms and for negotiation procedures.

7. 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, explicit connection setup and teardown, reliable feature negotiation, and reliable congestion feedback.

- 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. (Applications could still make the decision not to use ECN for a particular session.)

- 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 resistant to spoofed connection attacks ("SYN floods").

- Connection setup and teardown must use explicit handshakes, facilitating transmission through stateful firewalls.

In 2002, there was judged to be a consensus about the need for a new unicast transport protocol for unreliable datagrams, and the next step was then the consideration of more detailed architectural issues.

8. Security Considerations

There are no security considerations for this document. It does discuss a number of security issues in the course of problem analysis, such as DoS resistance and firewall traversal. The security considerations for DCCP are discussed separately in [RFC4340].

9. Acknowledgements

We would like to thank Spencer Dawkins, Jiten Goel, Jeff Hammond, Lars-Erik Jonsson, John Loughney, Michael Mealling, and Rik Wade for feedback on earlier versions of this document. We would also like to thank members of the Transport Area Working Group and of the DCCP Working Group for discussions of these issues.

Informative References

[BRS99] Balakrishnan, H., Rahul, H., and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts", SIGCOMM, Sept. 1999.

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

[PF01] Padhye, J. and S. Floyd, "Identifying the TCP Behavior of Web Servers", SIGCOMM 2001.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[RFC2525] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP Implementation Problems", RFC 2525, March 1999.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3448] Handley, M., Floyd, S., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 3448, January 2003.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

[RFC3714] Floyd, S. and J. Kempf, "IAB Concerns Regarding Congestion Control for Voice Traffic in the Internet", RFC 3714, March 2004.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

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

Authors' Addresses

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

EMail: floyd@icir.org

Mark Handley
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
UK

EMail: M.Handley@cs.ucl.ac.uk

Eddie Kohler
4531C Boelter Hall
UCLA Computer Science Department
Los Angeles, CA 90095
USA

EMail: kohler@cs.ucla.edu

Full Copyright Statement

Copyright (C) The Internet Society (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

Appendix B

Acronyms

AIMD Additive increase/multiplicative decrease 16

API Application Programming Interface 38

B Bi-predictive Picture 37

BIC Binary Increase Congestion control 17

CCID Congestion Control Identifier 21

codec coder-decoder 10

CSV Comma Separated Values 56

DCCP Datagram Congestion Control Protocol 4

diffserv Differentiated Services 12

DNS Domain Name System 18

e2e end-to-end 12

ECN Explicit Congestion Notification 10

FEC Forward Error Correction 24

FIFO First In First Out 6

HTTP Hypertext Transfer Protocol 11

I Intra-coded Picture 37

ICIR ICSI Center for Internet Research 44

ICMP Internet Control Message Protocol 14

IETF Internet Engineering Task Force 5

IP Internet Protocol 9

IPTV IP Television 120

ISP Internet Service Provider 20

ITU International Telecommunication Union 5

LIFO Last In First Out 5

MFRC Media Friendly Rate Control 23

MOS Mean Opinion Score 44

MPEG Moving Pictures Expert Group 27

MTP Multimedia Transmission Protocol 27

NAPT Network Address and Port Translation 19

NTP Network Time Protocol 57

P Predicted Picture 37

QoS Quality of Service 2

RTCP Real-time Transport Control Protocol 19

RFC Request for Comments 6

RTP Real-time Transport Protocol 19

RTT Round Trip Time 5

RTW-DCCP Run-Time adjusted Window-based DCCP 22

SBPN Send the Best Packet Next 4

SCTP Stream Control Transmission Protocol 20

SMTP Simple Mail Transport Protocol 11

SSVP Scalable Streaming Video Protocol 27

TCP Transmission Control Protocol 13

TCP/IP TCP over IP 1

TFRC TCP Friendly Rate Control 22

TLCTCP Time-lined TCP 27

UDP User Datagram Protocol 2

VoIP Voice over IP 2

VTP Video Transport Protocol 27

WHDI Wireless Home Digital Interface 28

References

[1] Network Simulator. http://www.isi.edu/nsnam/ns.

[2] TCP Probe. http://www.linuxfoundation.org/.

[3] Wireless Home Digital Interface. http://www.whdi.org.

[4] IETF 68 Proceedings. http://www.ietf.org/proceedings/68/, 2007.

[5] MSN Messenger. http://msn.messenger.com, Accessed 2009.

[6] Internet World Stats. http://www.internetworldstats.com/stats.htm, Accessed 2011.

[7] tcpdump. http://www.tcpdump.org, Accessed 2012.

[8] Cisco Visual Networking Index: Forecast and Methodology, 2011-2016 Internet World Stats. http://www.cisco.com, Accessed 2013.

[9] Ekiga. http://ekiga.org, Accessed 2013.

[10] Hulu. http://www.hulu.com, Accessed 2013.

[11] Netflix. http://www.netflix.com, Accessed 2013.

[12] Skype. http://www.skype.com, Accessed 2013.

[13] X Window System. http://www.x.org, Accessed 2013.

[14] Youtube. http://www.youtube.com, Accessed 2013.

[15] Toufik Ahmed, Ahmed Mehaoua, Raouf Boutaba, and Youssef Iraqi. IP Video Streaming With Fine-Grained TCP-Friendly Rate Adaptation. In *Proceeding of the IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS)*, Belfast, Northern Ireland, 2003.

[16] R. Allman, V. Paxson, and E. Blanton. RFC5681: TCP Congestion Control, 2009.

[17] Paul D. Amer, Christophe Chassot, Thomas J. Connolly, Michel Diaz, and Phillip Conrad. Partial-order Transport Service for Multimedia and Other Applications. *IEEE/ACM Transactions on Networking*, 2(5):440–456, 1994.

[18] Goel Ashvin, Krasic Charles, Li Kang, and Walpole Jonathan. Supporting Low Latency TCP-Based Media Streams. In *Proceedings of IWQoS*, Miami Beach, Florida, USA, 2002.

[19] Alex Balk, Dario Maggiorini, Mario Gerla, and M. Y. Sanadidi. Adaptive MPEG-4 Video Streaming with Bandwidth Estimation. In *Quality of Service in Multiservice IP Networks: Second International Workshop, QoS-IP 2003*, pages 525–538, Milano, Italy, 2003.

[20] S. Bhandarkar, A. L. N. Reddy, and M. Allman. RFC4653: Improving the Robustness of TCP to Non-Congestion Events, August 2006.

[21] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC2475: An Architecture for Differentiated Services, 1998.

[22] Jean-Chrysostome Bolot and Thierry Turletti. A Rate Control Mechanism for Packet Video in the Internet. In *INFOCOM (3)*, pages 1216–1223, 1994.

[23] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[24] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.

[25] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *SIGCOMM*, pages 14–26, 1992.

[26] R.J. Cole and J. Rosenbluth. Voice over IP Performance Monitoring. *ACM Computer Communication Review (CCR)*, vol 31(no 2):pages 9–24, 2001.

[27] Min Dai and Dmitri Loguinov. Analysis of rate-distortion functions and congestion control in scalable internet video streaming. In *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 60–69, New York, NY, USA, 2003. ACM Press.

[28] Philippe de Cuetos and Keith W Ross. Unified Framework for Optimal Video Streaming. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1479–1489, Hong Kong, 2004. IEEE.

[29] Giuseppe D. De Marco, Maurizio Longo, and Fabio Postiglione. Runtime Adjusted Congestion Control for Multimedia: Experimental Results. In *AINA '04: Proceedings of the 18th International Conference on Advanced Information Networking and Applications Volume 2*, Washington, DC, USA, March 2004. IEEE Computer Society.

[30] L. Eggert and G. Fairhurst. RFC5405: Unicast UDP Usage Guidelines for Application Designers, November 2008.

[31] Lars Eggert, Horia V. Balan, Saverio Niccolini, and Marcus Brunner. An Experimental Evaluation of Voice Quality over the Datagram Congestion Control Protocol. In *Infocomm 2007*, 2007.

[32] Chip Elliot. High-quality multimedia conferencing through a long-haul packet network. In *MULTIMEDIA '93: Proceedings of the first ACM international conference on Multimedia*, pages 91–98, New York, NY, USA, 1993. ACM Press.

[33] N. Feamster, D. Bansal, and H. Balakrishnan. On the Interactions Between Layered Quality Adaptation and Congestion Control for Streaming Video. *11th International Packet Video Workshop (PV2001), Kyongiu, Korea*, April 2001.

[34] W. Feng, M. Liu, B. Krishnaswami, and A. Prabhudev. A Priority-Based technique for the Best-Effort Delivery of Stored Video. *Proc. of Multimedia Computing and Networking*, pages 286–300, 1999.

[35] S. Floyd and M. Allman. RFC5033: Specifying New Congestion Control Algorithms, August 2007.

[36] S. Floyd, M. Allman, I. Jain, and P. Sarolahti. RFC4782: Quick-Start for TCP and IP, January 2007.

[37] S. Floyd, M. Handley, and E. Kohler. RFC4336: Problem Statement for the Datagram Congestion Control Protocol (DCCP), March 2006.

[38] S. Floyd and E. Kohler. RFC4341: Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control, March 2006.

[39] S. Floyd and E. Kohler. RFC5622: Profile for DCCP Congestion Control ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP), August 2009.

[40] S. Floyd, E. Kohler, and J. Padhye. RFC4342: Profile for DCCP Congestion Control ID 3: TFRC Congestion Control, March 2006.

[41] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE ACM Transactions on Networking*, 7(4):458–472, 1999.

[42] Cheng P. Fu and Soung C. Liew. TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks. *IEEE Communications*, 21(2):216–228, 2003.

[43] H. Gharavi and K. Ban. Rate adaptive video transmission over ad-hoc networks. *Electronics Letters*, 40(19):1177–1178, September 2004.

[44] F. Gont. RFC6633: Deprecation of ICMP Source Quench Messages, May 2012.

[45] Lei Guo, Enhua Tan, Songqing Chen, Zhen Xiao, Oliver Spatscheck, and Xiaodong Zhang. Delving into Internet Streaming Media Delivery: A Quality and Resource Utilization Perspective. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 217–230, New York, NY, USA, 2006. ACM Press.

[46] S. Ha, I. Rhee, and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[47] M. Handley, S. Floyd, J. Padhye, and J. Widmer. RFC3448: TCP Friendly Rate Control (TFRC): Protocol Specification, January 2003.

[48] M. Handley, S. Floyd, J. Padhye, and J. Widmer. RFC5348: TCP Friendly Rate Control (TFRC): Protocol Specification, September 2008.

[49] Jeffrey A. Hart. The Net Neutrality Debate in the United States. *Journal of Information Technology and Politics*, 8(4):418–443, 2011.

[50] S. Hemminger. Netem - Emulating Real Networks in the Lab. *Proc. Linux Conference Australia*, 2005.

[51] J Hruska. Industry group working on yet another wireless HD standard. http://arstechnica.com/hardware/news/2008/07/industry-group-working-on-yet-another-wireless-hd-standard.ars, Accessed 2008.

[52] B. Huffaker, M. Fomenkov, D. Moore, and K. Claffy. Macroscopic analyses of the infrastructure: Measurement and visualization of Internet connectivity and performance. In *Passive and Active Network Measurement Workshop (PAM)*, Amsterdam, Netherlands, April 2001. RIPE NCC.

[53] ICIR DCCP Implementation. http://www.icir.org/kohler/dcp/linux-2.4.20-dccp-modifications.tgz, Accessed 2005.

[54] Information Sciences Institute. RFC793: Transmission Control Protocol, 1981.

[55] Information Sciences Institute. RFC2309: Recommendations on Queue Management and Congestion Avoidance in the Internet, 1998.

[56] NLANR/DAST : Iperf - The TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/Projects/Iperf/, Accessed 2007.

[57] ITU-T. Pulse code modulation (PCM) of voice frequencies. Technical report, ITU-T Recommendation G.711, November 1988.

[58] ITU-T. One-way transmission time. Technical report, ITU-T Recommendation G.114, May 2003.

[59] ITU-T. The E-Model, a computational model for use in transmission planning. Technical report, ITU-T Recommendation G.107, March 2005.

[60] ITU-T. Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP). Technical report, ITU-T Recommendation G.729, December 2006.

[61] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

[62] W. John and S. Tafvelin. Analysis of internet backbone traffic and header anomalies observed. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 111–116. ACM, 2007.

[63] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society*, volume 49, 1998.

[64] E. Kohler, S. Floyd, and A. Sathiaseelan. Faster Restart for TCP Friendly Rate Control (TFRC), draft 03. Work in progress. http://www.ietf.org/internet-drafts/draft-ietf-dccp-tfrc-faster-restart-03.txt, 2007.

[65] E. Kohler, M. Handley, and S. Floyd. Designing DCCP: Congestion Control Without Reliability. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 27–38. ACM, 2006.

[66] E. Kohler, M. Handley, and S. Floyd. RFC4340: Datagram Congestion Control Protocol (DCCP), March 2006.

[67] Charles Krasic, Jonathan Walpole, and Wu-Chi Feng. Quality-adaptive Media Streaming by Priority Drop. In *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 112–121, New York, NY, USA, 2003. ACM Press.

[68] Jim Kurose. Open Issues and Challenges in Providing Quality of Service Guarantees in High-Speed Networks. *ACM Computer Communication Review*, 23(1):6–15, 1993.

[69] J. Lai and E. Kohler. Efficiency and late data choice in a user-kernel interface for congestion-controlled datagrams. In S. Chandra and N. Venkatasubramanian, editors, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5680 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 136–142, December 2004.

[70] Averill M. Law. Practical Statistical Analysis of Simulation Output Data: The State of the Art. In R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, editors, *Winter Simulation Conference*, pages 67–72, Washington, DC, USA, December 2007. Winter Simulation Conference.

[71] Lulea. DCCP Projects. http://www.dccp.org/, Accessed 2005.

[72] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *IMC '09*, volume 9, pages 4–6, New York, USA, 2009.

[73] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297, New York, NY, USA, 2001. ACM Press.

[74] I. McDonald. DCCP Patches to Linux Kernel 2.6.20. http://wand.net.nz/~iam4/dccp/patches20/, 2007.

[75] I. McDonald and R. Nelson. Congestion control advancements in Linux. *Proc. Linux Conference Australia*, 2006.

[76] Patrick McManus. DCCP Implementation by Patrick McManus. http://www.ducksong.com:81/dccp/, Accessed 2006.

[77] Alberto Medina, Mark Allman, and Sally Floyd. Measuring the evolution of transport protocols in the internet. *SIGCOMM Comput. Commun. Rev.*, 35(2):37–52, April 2005.

[78] Arnaldo C. Melo. TCPfying the Poor Cousins. In *Ottawa Linux Symposium*, July 2004.

[79] Arnaldo C. Melo. DCCP on Linux. In *Ottawa Linux Symposium*, pages 305–311, 2005.

[80] B. Mukherjee and T. Brecht. Time-lined TCP for the TCP-friendly Delivery of Streaming Media. *International Conference on Network Protocols (ICNP), Osaka Japan*, pages 165–176, 2000.

[81] John Nagle. Congestion Control in IP/TCP Internetworks. *SIGCOMM Comput. Commun. Rev.*, 14(4):11–17, October 1984.

[82] NIST Net Home Page. http://snad.ncsl.nist.gov/itg/nistnet/, Accessed 2005.

[83] Paul Njoroge, Asuman Ozdaglar, Nicols E. Stier-moses, and Gabriel Y. Weintraub. Investment in Two Sided Markets and the Net Neutrality Debate. *Columbia Business School DRO (Decision, Risk and Operations) Working Paper*, (2010-05), 2010.

[84] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, New York, NY, USA, 1998. ACM Press.

[85] Fernando Paganini, Zhikui Wang, John C. Doyle, and Steven H. Low. Congestion control for high performance, stability, and fairness in general networks. *IEEE/ACM Trans. Netw.*, 13(1):43–56, February 2005.

[86] Panagiotis Papadimitriou and Vassilis Tsaoussidis. SSVP: A congestion control scheme for real-time video streaming. *Computer Networks*, 51(15):4377–4395, October 2007.

[87] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Netw.*, 1(3):344–357, June 1993.

[88] C. Parsa and JJ Garcia-Luna-Aceves. Differentiating congestion vs. random loss: a method for improving TCP performance over wireless links. In *Wireless Communications and Networking Conference, 2000. WCNC. 2000 IEEE*, volume 1, pages 90–93. IEEE, 2000.

[89] Vern Paxson. End-to-end Internet packet dynamics. *IEEE ACM Transactions on Networking*, 7(3):277–292, 1999.

[90] T. Phelan. Strategies for Streaming Media Applications Using TCP-Friendly Rate Control. http://tools.ietf.org/id/draft-ietf-dccp-tfrc-media-00.txt.

[91] T. Phelan. Media Friendly Rate Control (MFRC). http://www.phelan-4.com/dccp/draft-phelan-mfrc-00.txt, July 2004.

[92] T. Phelan. Datagram Congestion Control Protocol (DCCP) User Guide. Work in progress. http://tools.ietf.org/id/draft-ietf-dccp-user-guide-03.txt, 2005.

[93] T. Phelan, G. Fairhurst, and C. Perkins. DCCP-UDP: A Datagram Congestion Control Protocol UDP Encapsulation for NAT Traversal, November 2012.

[94] J. Postel. RFC768: User Datagram Protocol, 1980.

[95] J. Postel. RFC792: Internet Control Message Protocol, September 1981.

[96] K. Ramakrishnan and S. Floyd. RFC2481: A Proposal to Add Explicit Congestion Notification (ECN) to IP, 1999.

[97] R. Rejaie and A. Reibman. Design Issues for Layered Quality-Adaptive Internet Video Playback. *Lecture Notes in Computer Science*, 2170:433+, 2001.

[98] Reza Rejaie, Mark Handley, and Deborah Estrin. Quality Adaptation for Congestion Controlled Video Playback Over the Internet. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, volume 29, pages 189–200, New York, NY, USA, October 1999. ACM Press.

[99] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-End Arguments in System Design. In *Proceedings of the Second International Conference on Distributed Computing Systems*, pages 509–512, 1981.

[100] Angela Sasse, Ulf Bilting, Claus D. Schulz, and Thierry Turletti. Remote Seminars through Multimedia Conferencing: Experiences from the MICE Project. In *Proc. INET'94/JENC5*, pages 251/1–251/8, Prague, June 1994.

[101] A. Sathiaseelan and G. Fairhurst. TCP Friendly Rate Control (TFRC) for Bursty Media Flows. *Computer Communications*, 2011.

[102] Henning Schulzrinne, Steven Casner, R Frederick, and Van Jacobson. RFC3550: RTP: A Transport Protocol for Real-Time Applications. 2003.

[103] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. RFC3530: Network File System (NFS) version 4 Protocol, April 2003.

[104] P . Srisuresh and K. Egevang. RFC3022: Traditional IP Network Address Translator (Traditional NAT), March 2001.

[105] Richard W. Stevens. *The Protocols (TCP/IP Illustrated, Volume 1)*. Addison-Wesley Professional, December 1993.

[106] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, T. Rytina, M. Kalla, L. Zhang, and V. Paxson. RFC2960: Stream Control Transmission Protocol.

[107] Vassilios Tsaoussidis and Songbin Wei. Reliability / Throughput / Jitter Tradeoffs for Real-time Transport Protocols. Technical report, IEEE, 1999.

[108] J. M. Valin, T. B. Terriberry, C. Montgomery, and G. Maxwell. A High-Quality Speech and Audio Codec With Less Than 10-ms Delay. *Trans. Audio, Speech and Lang. Proc.*, 18(1):58–67, January 2010.

[109] WAND. WAND implementation of DCCP. http://research.wand.net.nz/software/dccp.php, Accessed 2006.

[110] Zhiheng Wang, Sujata Banerjee, and Sugih Jamin. Media-friendliness of a slowly-responsive congestion control protocol. In *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 82–87, New York, NY, USA, 2004. ACM Press.

[111] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *IEEE Infocom 2004*, 2004.