An Experimental Evaluation of Voice-over-IP Quality over the Datagram Congestion Control Protocol

School of Engineering and Science
International University Bremen
May 2006

Vlad Balan

Review Committee: Jürgen Schönwälder
Lars Eggert

I hereby certify that the current thesis is independent work that has not been submitted elsewhere.

Abstract

The presence of voice-over-IP traffic in the Internet is constantly increasing, as it offers improved connectivity and quality-of-service comparable to classical telephony at costs that the rival circuit-switched telephony systems cannot match. This paper evaluates the audio quality of voice-over- IP calls that use the Datagram Congestion Control Protocol (DCCP), a congestion-controlled alternative for the User Datagram Protocol (UDP) that carries most voice-over-IP calls today. A framework for assessing the impact of the transport layer on VoIP streams is presented. The experimental results illustrate some of the problems that the proposed congestion control methods face in preserving the perceived quality of VoIP transmission. The contributions of the current thesis are twofold. It evaluates the performance of the currently proposed congestion control methods in transporting VoIP traffic, and tries to illustrate some of their possible shortcomings. It also establishes a testing methodology for evaluating the impact of transport protocols on VoIP quality.

Contents

Acknowledgements

I would like to thank Lars Eggert, Saverio Niccolini and Marcus Brunner at NEC as well as prof. Jürgen Schönwälder at IUB for supervising this thesis and for their support during my work. I would like to thank Mr. Yoshifumi Nishida for his kind support in adapting the KAME DCCP implementation to the purposes of our experiments. Finally I would like to thank my family as well as my colleagues in Heidelberg and in Bremen for their support.

Chapter 1

Introduction

The presence of voice-over-IP traffic in the Internet is constantly increasing, because it offers improved connectivity and quality-of-service comparable to classical telephony at costs that the rival circuit-switched telephony systems cannot match. This paper evaluates the audio quality of voice-over- IP calls that use the Datagram Congestion Control Protocol (DCCP), a congestion-controlled replacement for the User Datagram Protocol (UDP) that carries most voice-over-IP calls today.

While the infrastructure of the Internet is adapting to the changing demands of the transported traffic, the design of its transmission protocols remains more appropriate for data streams than for multimedia or voice streams that require low-latency delivery. The key to the advancement of the new services is the ability to provide a decent perceived quality and to adapt to the rapid network growth.

For many years, provisioning has been the strategy of choice for improving the quality of offered services. Because many of the connections that will be used in the future as attractive replacements for classical telephony will be bandwidth-limited, partly due to a lack of available bandwidth in the case of many edge connection situated in developing areas, it seems that finding a solution for dealing with congestion for streams having tight delivery requirements can no longer be avoided.

The slow adoption of technologies such as IPv6, IPSec or ECN has proven that modifications of the core network architecture are difficult to achieve in a reasonable time frame. As described by the IAB [1], while traffic management alternatives offering differentiated services have been developed and occasionally deployed on particular network segments, VoIP traffic over best-effort is expected to rise significantly as the wide-spread deployment of the Internet continues. In order to prevent a probable congestion control caused by media streams, end-to-end congestion control has to be extended for accommodating interactive multimedia streams.

DCCP is an alternative to the UDP transport protocol, prevalently used for multimedia-stream transmission, which adds customizable congestion-control capabilities and makes congestion occurrence visible to the upper-layer applications. Although TCP also provides congestion control, its methods are closely related to the reliable nature of the protocol, and to the bulk nature of the data transfers. By contrast, DCCP's congestion control mechanisms try to better

serve multimedia applications.

The purpose of the current work is to evaluate the impact that DCCP's different proposed congestion control methods have on the perceived quality of VoIP transmission. The methods analyzed are the TCP Friendly Rate Control (TFRC), as standardized for DCCP, TFRC for small packets and the faster restart variant of TFRC for small packets .

Chapter 2 gives an overview of the DCCP protocol, its currently defined congestion control methods, the KAME implementation used in the current experiments and the proposed APIs for interfacing DCCP with user-space applications.

Chapter 3 addresses some of the issues appearing when sending VoIP traffic over a congestion-aware protocol. It continues by giving an overview of the assessment methods of the perceived quality of voice transmission, also in the context of packet-switched networks. The chapter discusses some statistical properties of voice streams that might be used in trying to optimize the quality of the transmission. Finally, some strategies for dealing with congestion build-up in the case of variable rate codecs are presented.

Chapter 4 presents the experimental setup and the metrics involved in the quality evaluations. The chapter begins with a description of the conversation model used for generating speech patterns, followed by the presentation of the metric used in evaluating the way in which packet transmission reflects in voice quality. Next, it presents an algorithm for bounding the performance of playout buffers, adapted for fitting the quality estimation requirements. In the end, it introduces the characteristics of the experimental setup.

Chapter 5 proceeds to the experimental results and discusses their significance. It presents an overview of the main conclusions derived from the experiments performed. The performance of the various variants of TFRC is compared to the one of UDP and TCP, and the reasons affecting it are shortly discussed.

Chapter 6 summarizes the experiments and methods presented throughout the thesis and derives conclusions related to the research question. It also indicates some possible future research themes related to the quality of VoIP transmission over DCCP.

The contributions of the current thesis are twofold. It evaluates the performance of the currently proposed congestion control methods in transporting VoIP traffic, and tries to illustrate some of their possible shortcomings. It also establishes a testing methodology for evaluating the impact of transport protocols on VoIP quality.

Chapter 2

The DCCP Protocol

The current chapter gives an overview of the DCCP protocol, its currently defined congestion control methods and of the implementation used in the current experiments. Two proposed APIs are discussed.

2.1 Protocol Description

The Datagram Congestion Control Protocol (DCCP) is a transport protocol oriented towards the delivery of unreliable datagrams. The main design objective and extension over the traditional UDP protocol was to assure congestion control for its datagram flows.

DCCP has a modular design, separating the core functionality of the protocol from the implementation of the congestion control mechanism.

The main features of the core protocol are, as stated in [2]:

An unreliable flow of datagrams, with acknowledgments - Just like UDP datagrams, DCCP datagrams are not subject to retransmission, the protocol being intended for applications for which the timely transmission of data is more important than its overall consistency. Since the majority of congestion control methods relies on acknowledgments of sent data packets, the extension to the classical UDP send-and-forget scheme is that each sent packet must be acknowledged. The actual form of the acknowledgment can be chosen by the congestion control method from individual or piggy-backed acknowledgment packets or acknowledgment vectors sent as DCCP options.

Reliable handshakes for connection setup and teardown - DCCP is a connection-oriented protocol, in order to allow better interaction with middleboxes (firewalls and NATs will be able to detect the establishment of the connection and better map the traffic). The initial phase of the connection is a three-way handshake, involving possibly feature negotiation and a cookie to be returned by the client. DCCP introduces the concept of services (32-bit Codes) that identify the application-level service to which the client is trying to connect. Note that in TCP or UDP a similar role is fulfilled by the port numbers.

Mechanisms allowing servers to avoid holding state for unacknowledged connection attempts and already-finished connections - During the initialization phase of the connection the server generates a cookie intended for the server to avoid generating state until the three-way handshake has completed.

During the connection-termination phase, or at any point when a protocol malfunction-operation is detected, a party can finish a connection using a DCCP-Reset packet, and discard all state associated with the connection.

Congestion control incorporating Explicit Congestion Notification (ECN) [3] and the ECN Nonce [4] - DCCP implementations are ECN-aware, and in most situations treat ECN marked packets similarly to dropped packets in computing the modifications to the transmit rate.

Acknowledgment mechanisms communicating packet loss and ECN information - DCCP acknowledgments report lost or ECN-marked packets, corrupt or dropped data. Packets can be acknowledged through acknowledgments packets generated at a certain Ack Ratio of the received packets, or through Acknowledgment vector option, which store run-length encoded acknowledgment information. Mechanisms have been devised in order to insure the consistency of acknowledgment packets (a source for inconsistency could be packet duplication).

Congestion Control Mechanisms may use additional information such as time stamps in computing the transmit rate.

Optional mechanisms for informing the application of congestion events.

Path Maximum Transmission Unit (PMTU) discovery - DCCP provides PMTU support, and a DCCP implementation should prevent applications from writing datagrams that will exceed the MTU, unless the application has requested that fragmentation should be performed.

A choice of modular congestion control mechanisms Applications have the possibility to choose the preferred congestion control mechanisms.

The protocol has the following main components:

Packet types The packet types of the protocol are:

- DCCP-Request initiates a connection
- DCCP-Response sent by the server in response to a DCCP-Request
- DCCP-Data used to transmit data
- DCCP-Ack used to transmit pure acknowledgments
- DCCP-DataAck used to transmit data with piggybacked acknowledgments
- DCCP-CloseReq used by the server to request that the client close the connection
- DCCP-Close used by the client or the server to close the connection; triggers a DCCP-Reset in response
- DCCP-Reset used to terminate the connection, either normally or abnormally
- DCCP-Sync, DCCP-SyncAck used to re-synchronize sequence numbers after large bursts of losses

Sequence numbers - DCCP packets carry sequence numbers, to make possible the identification and reporting of lost packets. DCCP sequence numbers increment by one per packet, and every packet, no matter of its type, increments the sequence number, allowing DCCP to detect all packet loss.

States DCCP defines the following states:

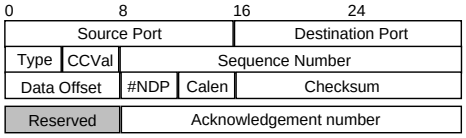

Figure 2.1: DCCP - a generic header and an acknowledgment [5].The packet formats are described by the DCCP draft.

- CLOSED represents nonexistent connections.
- LISTEN represents server sockets in the passive listening state
- REQUEST a client socket enters this state, from CLOSED, after sending a DCCP-Request in order to start a connection
- RESPOND a server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client
- PARTOPEN a client socket enters this state after receiving a DCCP-Response from the server. After entering this state, the client must include acknowledgments in its packets, and can start sending data packets.
- OPEN the data transfer option of a DCCP connection, applying to both client and server
- CLOSEREQ a server socket enters this state to signal that the connection is over, and the client should enter TIMEWAIT
- CLOSING this state applies to both client and server, and they enter it in order to close the connection
- TIMEWAIT Server or client sockets remain in this state 2MSL (4 minutes) in order to prevent mistakes due to delivery of old packets. Only one connection endpoint needs to enter this state, and the server can use the CLOSEREQ packet in order to ask the client to enter it.

The state diagram of the protocol is provided for both the client and the server.

Congestion Control - DCCP's main design purpose was to offer efficient congestion control for multimedia streams. To this end, support for implementing different congestion control schemas was implemented, from which applications can choose. Each congestion control mechanism is identified by a CCID, and the communicating parties agree through feature-negotiation on the mechanism to be used. Currently two mechanisms have been standardized:

- TCP Like Congestion Control (CCID 2) implements congestion control through tracking a transmission window, and regulating the transmit rate similarly to TCP.

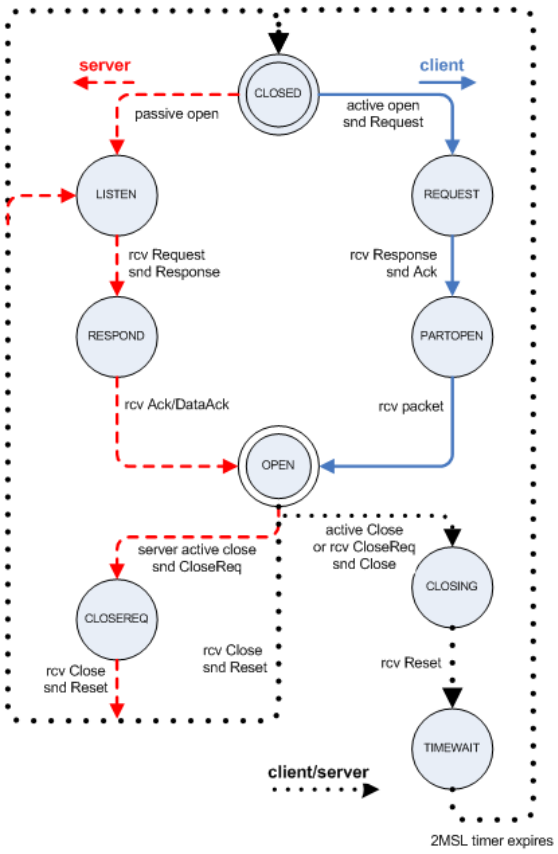

Figure 2.2: DCCP State Diagram [6]

- TCP Friendly Congestion Control (TFRC) (CCID 3) implements congestion control by tracking the rate at which packets are lost (but at most one packet per RTT), and varies the transmit rate in a smoother manner, using additive increases and subtractive decreases.

The behavior of the respective CCIDs are described in separate documents [7] [8].

The two half-connection comprising a DCCP connection can be governed by different congestion control mechanisms.

Features - DCCP features are connection attributes upon which the two end-points agree, and can be referring to one connection endpoint. Their value is negotiated through the use of the DCCP-Change and DCCP-Confirm packet options.

The DCCP draft [2] provides the following example of a DCCP connection:

```
Client                                                  Server
------                                                  ------
                    (0) No connection
CLOSED                                                  LISTEN

                    (1) Initiation
REQUEST       DCCP-Request -->
                              <-- DCCP-Response         RESPOND
PARTOPEN      DCCP-Ack or DCCP-DataAck -->

                          (2) Data transfer
OPEN           <-- DCCP-Data, Ack, DataAck -->          OPEN

                    (3) Termination
                              <-- DCCP-CloseReq         CLOSEREQ
CLOSING       DCCP-Close -->
                                 <-- DCCP-Reset         CLOSED
TIMEWAIT
CLOSED
```

Figure 2.3: DCCP connection example

2.2 Congestion Control - TFRC and Variants

Congestion occurrence was first observed during the early stages of the Internet [9] and rate regulating algorithms for data connections were introduced by Jacobson [10] [11].

The assumption behind this early class of congestion response methods is that the Internet is a black box, in which congestion occurs from time to time and can be detected through packet drops. These algorithms rely on TCP retransmission and are therefore intended for transfers of bulk of data, where momentary rate decreases and delays due to retransmissions do not influence significantly the quality of service. More exactly, upon detecting congestion

through packet drops, TCP will halve the transmit rate and retransmit the affected packets.

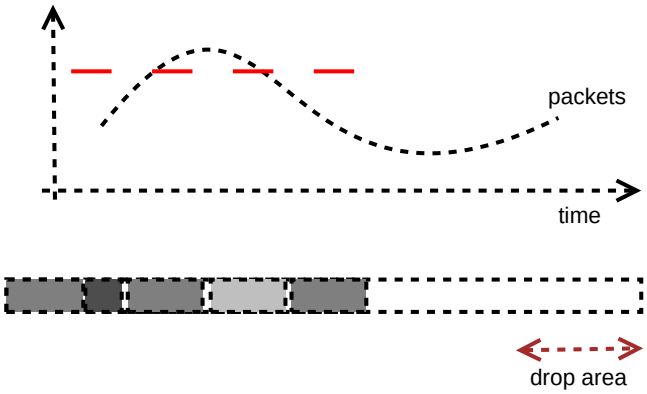

Figure 2.4: Simple Drop-Tail Queue

For interactive applications, relying on the low latency of the received data, such a behavior is more problematic. Application developers would rather adopt a more conservative sending rate than start facing congestion through packet drops.

By contrast, a more modern class of congestion control methods tries to preempt the congestion behavior and has to take into account the particularities of the network devices involved in transmission and congestion control. Routers have to provide support for such congestion detection and avoidance techniques, and the transport layer protocols must act appropriately upon receiving a congestion notification. Some of these protocols use the IP layer for transporting congestion notifications, although this raises some problems in some environments such as tunneled connections.

As discussed in [12] the typical router maintains a packet queue for all flows and drops only those packets for which the queue does not have available space, as illustrated in Figure 2.4. The result of this is that the queue can be for a long time in an almost filled size because flows might try to aggressively maintain a large data transmit rate and periodically overflow the queue. This behavior might seem fair; however we cannot ignore the importance of packet bursts in Internet data transmission. A burst of packets will need a significant part of the queue size, and if one of them will be dropped (very probable with a filled queue) the connection will have to back off.

The RFC 2309 [12] presents the queue management algorithm Random Early Detection (RED) devised to overcome this shortcoming of simple rate control. RED divides the queue occupancy levels in thee zones:

- a safe zone, for which all packets are sent further
- a transition zone, for which packets are dropped probabilistically, with a probability depending linearly on the level of occupancy of this zone
- a drop-zone, for which all packets will be dropped

The occupancy level of the queue (the average queue size) is computed using a an exponentially weighted low-pass filter which has the role of allowing bursts of traffic, since the average levels over time. The behavior of the RED queue is illustrated in Figure 2.5. Packets from the same burst (corresponding to the same flow) are more likely to receive the same marking, reducing thereby the number of different flows interrupted.

This congestion detection algorithm addresses the following issues in an efficient way:

- allowing bursts of traffic
- avoidance of global synchronization, that is of a simultaneous requirement for all flows to reduce their congestion window in the case of TCP
- enforcing a low-latency transmission (due to reduced average queue size)
- independence from the flow behavior

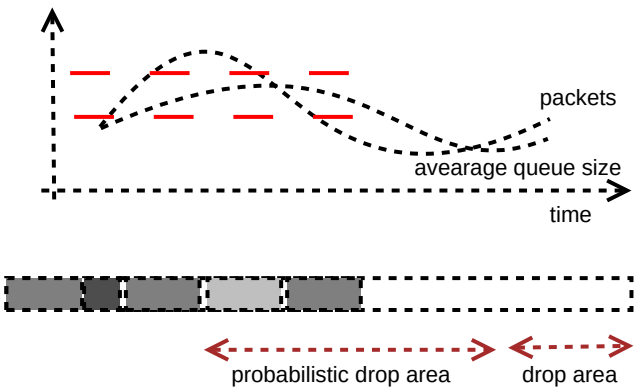

Figure 2.5: A RED Queue

Another problem RED and other congestion control methods face is the different, possibly aggressive behavior of some flows (for example unregulated UDP flows). Algorithms which maintain different queues for different traffic classes or flows (CBQ or FQ) are necessary in this situation, otherwise a connection lock-out might occur (a monopolization of the queue by one certain flow). Under Linux, Generalized RED (GRED) provides such a differentiation of flows [13].

Note that fragmentation of datagrams does not cope well with some packet loss measurement techniques, since a packet loss translates into a whole datagram loss.

Note: In order for it to function properly a RED queue's parameters should probably be fixed after considering statistical data collected from monitoring the traffic through the respective router.

2.2.1 TCP Reno congestion control

Some of the widely deployed TCP implementations in the Internet are based on the 4.3BSD-Reno distribution, and its congestion control method is often referred to as TCP Reno congestion control.

TCP Reno's transmission window size W regulates the rate of transmission. Initially the system starts with a default window size. As long as there is no indication of congestion, the window size is steadily increased with a factor of $1/W$ for each received acknowledgment. The receiver sends an acknowledgment for every b packets received, where for most TCP implementations $b = 2$. Therefore the rate of transmission increases linearly in time at a slope of $1/b$, as illustrated in Figure 2.7.

The sender perceives packet loss either by the reception of "triple-duplicate" acknowledgments (four packets having the same sequence number) or via timeouts. In the presence of duplicate acknowledgments, the sender halves the transmit rate, while upon perceiving a timeout, the sender reduces the transmit rate to one packet per time T_0, and if timeouts repeat, the retransmission time is doubled up to a maximal value of $64T_0$.

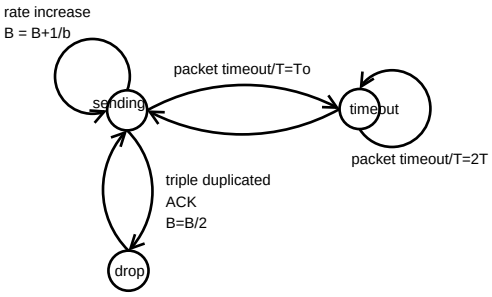

Figure 2.6: TCP-Reno states and actions

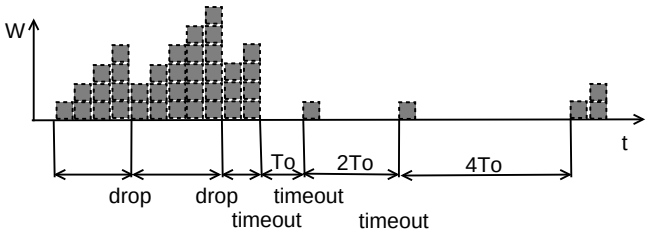

Figure 2.7: TCP-Reno rate control as presented in [14]

Since any congestion-controlled protocol sharing a line with TCP flows is supposed to achieve TCP-fairness (i.e., to have a similar transfer rate over different time intervals) different models that try to predict the rate achieved by

the Reno congestion control method under specific network conditions have been developed. At least one of these models has been investigated not only theoretically, but also compared with extensive observations of the behavior of connections between hosts situated at different distances on the Internet.

[15] presents a stochastic, Markov-based model of TCP-Reno's behavior under a specific packet loss rate. Due to its complexity the complete model is suitable for simulation but does not seem to have a closed solution. However, under specific assumptions the author obtains an accurate formula for the data rate of a TCP connection:

$$X(p) = \frac{s}{RTT\sqrt{2bp/3} + T_0(3\sqrt{3bp/8})p(1+32p^2)}$$

where

- X is the transmit rate in bytes/second
- s is the packet size in bytes
- RTT is the round trip time in seconds
- p is the loss event rate, between 0 and 1, of the number of loss events as a fraction of the number of packets transmitted
- T_0 is the TCP retransmission time in seconds
- b is the number of packets acknowledged by a single TCP acknowledgment

The transmit rate cannot exceed the value $\frac{W_{max}}{RTT}$ where W_{max} is the maximal TCP window size.

Note: TCP-SACK (TCP with selective acknowledgments) congestion control is becoming the more popular choice for congestion control in modern systems also due to the advance of wireless communication, in which occasional losses of packets are frequent but do not necessarily indicate congestion. The Linux and FreeBSD TCP stack implementations also deviate from the TCP-New Reno specification and implement TCP-SACK. For a detailed summary of the implementation differences between 4.3BSD-Reno and the Linux stack please consult [16].

2.2.2 TCP Friendly Rate Control

RFC 3448 [17] describes TFRC, a congestion control method designed to provided a smoother throughput of data more suited for multimedia applications. TFRC is further developed into a DCCP congestion control method in RFC 4342 [8]

TFRC uses the equation presented above in order to regulate its data throughput rate. The only parameters of the equation that must be estimated from measurements are the round trip time and the packet loss rate. Since RTT estimation is also common to TCP, in the following only the packet loss estimation process will be discussed.

Loss event rate measurement is performed at the receiver. Obtaining an accurate and stable measurement can assure that the transmit rate will not oscillate in a disruptive manner.

Assuming that all packets have unique sequence numbers, the receiver maintains a data structure which keeps track of which packets arrived and which are still missing, and includes also the receiver timestamp for received packets. Loss is detected by the arrival of at least three packets with sequence number higher than the one of the expected packet. Successive losses occurring during the same RTT period are considered a single loss event.

The missing packets create a loss history which is in turn translated into a series of loss intervals I_n , describing the time elapsed between the observed packet losses. The last eight loss intervals are added together with weighting factors decreasing in order to favor the most recent measurements, giving an average value I_{mean}:

$$I_{mean} = \sum_{i=0}^{7} w_i I_i$$

$$\text{where } w_i = (1.0, 1.0, 1.0, 1.0, 8, 0.6, 0.4, 0.2)$$

The most recent interval (the one having as upper bound the current time) is only taken into account if its presence would increase the I_{mean} value. Finally the value p is computed as

$$p = \frac{1}{I_{mean}}$$

The same RFC describes a history discounting mechanism that changes the weights of the computed mean value when the most recent loss interval is more than twice as large as the computed average loss interval. This mechanism is designed to provide a more aggressive behavior in the absence of congestion.

Before the first loss event occurs, the sender finds itself into a so-called slow-start mode. The length of the first loss interval is approximated by reversing the rate equation, using the number of bytes received during the last RTT (X_recv) before the initial loss occurrence as the left-hand side.

A DCCP stream operating under TFRC congestion control will face the following rate limitations:

- The transmission rate is upper bounded by X_calc, the computed maximal transmission rate
- The transmission rate is upper bounded by 2∗X_recv, the rate reported by the receiver through the last feedback packet; since feedback packets occur at most once per RTT, this assures that the transmission rate can at most double during one RTT.
- The transmission rate is lower bounded by s/R during the slowstart period, and by s/t_mbi during congestion avoidance, where t_mbi = 64s

2.2.3 TFRC for small packets

VoIP applications send small payload packets separated by fixed time intervals. [18] modifies the TFRC equation for dealing with this particular situation:

- the packet size s is fixed to 1500 bytes; TFRC streams sending small packets aim for achieving the same bandwidth utilization as a TCP stream using 1500 bytes-long segments.

- the send rate is multiplied by a factor $s\_true/(s\_true + H)$, with s_true denoting the average packet size and H the size of the packet headers (default value: 40 bytes).

2.2.4 Faster Restart for TFRC

VoIP application alternate periods of data sending with periods of idleness; at the beginning of a talkspurt, the application will try to reach the codec's nominal rate as fast as possible; if the congestion control mechanism adapts too slowly and the rate limitations cause the send buffer to fill up, the application will experience packet losses.

As mentioned earlier, TFRC can only double the transmission rate during one RTT; [19] allows the transmission rate to quadruple, as long as the network has proven in the past that it can sustain the target transmission rate; for this purpose a variable X_fast_max records the maximal rate achieved by the sender during the last loss-free period. The network rate is quadrupled while remaining under X_fast_max and doubled afterwards. The value of X_fast_max will be halved after each loss event detection.

The second modification brought by the faster restart variant is the setting of the initial transmission rate after an idle period to eight packets per RTT.

The current implementation, used in this project, emulates the faster restart behavior by allowing the sender to quadruple the rate at all times, based on the assumption that the safe rate X_fast_max will stabilize around the codec's nominal rate.

2.2.5 ECN Network Support

As mentioned earlier, packet dropping as a congestion notification method has the disadvantage of compromising the quality of service of certain applications, especially interactive ones, by disrupting the normal flow of data. For applications willing to maintain conservative transmit rates in exchange for obtaining increased reliability, an early warning would be preferable. Explicit Network Congestion (ECN) addresses exactly this problem.

The ECN defining document [3] states that two bits of the IP protocol header should be used by the in-path routers in order to signal a potential congestion condition. The second bit serves the purpose of providing a ECN nonce holder, which, over a longer period of time, insures the fair behavior of the communication partners.

The corrective action upon this notification is left to the transport protocol. Note that the ECN involves the intermediate routers (possibly using triggers similar to the ones coming from a RED queue), the IP protocol for carrying the notification and the nonce and the transport protocol for taking appropriate action.

RFC 3168 [3] describes some special implementation issues of the protocol, such as the fact that a congestion notification for a packet fragment at reassembly will trigger a congestion notification for the full packet, or that IP tunnels should mark embedded packets at decapsulation if congestion occurs during transport. Encrypted transports such as IPsec raise additional problems.

The success of ECN congestion control depends on its the degree of deployment in the routers, especially those sitting at the edge of the network where

congestion is likely to occur. Since ECN offer the possibility of regulating traffic without quality disruptions, its advance is linked to the expansion of interactive applications.

The open-source operating systems FreeBSD and Linux as well as the major vendors for routers have added ECN support. It is up to the system administrators to use this option properly for offering improved performance.

2.3 DCCP Mechanisms Implementation

The current section presents the DCCP implementation used in the experiments, and gives an overview of two of the proposed protocol APIs. The first API is based on shared memory zones between the kernel and the application, tries to achieve high performance and offers the possibility of late decision on the actual packets to be sent. The second API is based on the classic sockets interface, minimizing the complexity of porting applications to the new protocol.

2.3.1 The KAME implementation

The FreeBSD KAME kernel includes a DCCP protocol implementation based on the Lulea University DCCP for FreeBSD project [20]. This implementation is maintained by Yoshifumi Nishida.

The code implements currently most of the core DCCP functionality, including feature negotiation, and has a modular design which allows the addition of different CCID modules.

The implementation supports currently CCID 3 (TFRC), intended mainly for multimedia applications. The VoIP variant of CCID 3 will follow.

A number of networking tools have been patched to support the new protocol under this implementation (the socket API for DCCP is not yet standardized). These applications include:tcpdump, ethereal, iperf, netcat.

We have developed a ttcp clone, named kttcp, capable of reading framed data from the standard input and sending it over TCP, UDP or one of the DCCP congestion control modes to another endpoint of the network, where it will be printed on the standard output. The application is capable of gathering packet statistics.

The DCCP implementation has been extended during this project for supporting TFRC for small packets, TFRC for small packets with faster restart, and a modified version of the faster variant method. In order to comply with the requirements of interactive media traffic transmission, a sysctl handle for adjusting the size of the send buffer has been added.

2.3.2 The Packet-Ring socket API

Eddie Kohler and Junwen Lai have proposed a novel API for managing the transmission of unreliable datagrams using DCCP [21]. The API has as its strong-points high throughput, kernel-enforced congestion control and late data choice, i.e. the application can commit to sending a piece of data very late in the data sending process. The constructive design of the API encourages applications to assign priorities to the packets constructed and, in case congestion is detected, select the most important ones for transmission. This approach

matches the design of most modern codecs, for which certain frames (for example synchronization frames) have a much larger qualitative impact than others.

The classical sendmsg() API allows the kernel to perform buffering of packets to be sent, and such buffering is likely to occur under congestion. Applications might want to exchange this increase in reliability for timeliness of the delivery.

The solution proposed as an alternative is using a *packet ring* data structure, stored in memory shared by the application and the kernel, in a manner similar to DMA rings. The application queues packets for transmission by placing them at the end of the packet ring, transparent to the kernel.

For communication between the kernel and the application, four pointer variables are available: *dev_i, kern_i, umod_i, user_i*. Their relative order is fixed, and the kernel has control over *dev_i* and *kern_i* while the application has control over *umod_i* and *user_i*. *umod_i* marks the limit between the packets that the kernel can send and the ones that the application can still modify.

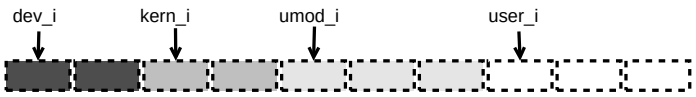

Figure 2.8: Packet ring

The benefits of using this API are double: not only does the application achieve faster transmission of datagrams, but in case of congestion the more important datagrams have a higher chance of making it through.

One could further develop this idea into a sendmsg()-like transmission function having a priority parameter for allowing the selection of packets to be transmitted in case of congestion. In this way the changes to the existing codebase currently running on top of UDP would be minimal.

2.3.3 User-space implementation specifics

An user-space application willing to use the DCCP protocol as its transport can invoke it through an adapted socket API. The steps to be taken for opening and using a DCCP connection are the following:

- *Filling out a hosthints structure(client).* The hosthints structure to be used by getaddrinfo() for a client must be filled with the values ai_socktype = SOCK_CONN_DGRAM and ai_protocol = IPPROTO_DCCP.
- *Setting the hostflags to AI_PASSIVE (server).*
- *Calling the getaddrinfo() function(client).*
  Note: The getaddrinfo() function does not yet support DCCP hints.
- *Initializing a socket.* The socket() call uses the hostinfo structure filled by getaddrinfo()
- *Binding the socket (client)*
- *Setting the CCID*
  int ccid; setsockopt(fd, IPPROTO_DCCP, DCCP_CCID, &ccid, sizeof(ccid))

- *Setting the maximum segment size option*
  int buflen; setsockopt(fd, IPPROTO_DCCP, DCCP_MAXSEG, &buflen, sizeof(buflen))

- *Setting the DCCP service option.* The role DCCP service option is analogous to the one of a port for a UDP or TCP socket. setsockopt(fd, IPPROTO_DCCP, DCCP_SERVICE, dccp_services, dccp_services_nr * sizeof(int))

- *Create a connection(client).* Since DCCP is connection-oriented, the connect() function call has been extended for DCCP.
  connect(fd, const struct sockaddr *name, socklen_t namelen)

- *Listen for a connection and accept incoming connections (server).*
  listen(fd, 0)
  fd = accept(fd, struct sockaddr * restrict addr, socklen_t * restrict addrlen)

- *Write to the socket or read from it.* The send() and recv() system calls allow writing to or reading from the socket.

- *Close the socket.* The socket should be closed using close(fd)

The current chapter presented the DCCP protocol, its currently proposed congestion control methods, and details of their implementations.

Chapter 3

The Impact of DCCP on VoIP

The current chapter addresses some of the issues appearing when sending VoIP traffic over a congestion-aware protocol. It continues by giving an overview of the assessment methods of the perceived quality of voice transmission, also in the context of packet-switched networks. The chapter discusses some statistical properties of voice streams that might be used in trying to optimize the quality of the transmission. Finally, some strategies for dealing with congestion build-up in the case of variable rate codecs are presented.

3.1 Strategies for VoIP DCCP Applications

The draft [22] presents some of the problems that applications using DCCP for VoIP telephony face. These problems originate in the following requirements of VoIP services:

- Low latency: VoIP applications should have an end-to-end latency of less than 150ms in order to present the user with a feeling of interactivity. Users are accustomed to such delays from using the normal circuit-switched phone service. While the latency is mainly generated by the network's behavior, an efficient strategy for VoIP applications should not increase it, nor should it add to it the problem of jitter (variations of the packet latency being perceived by the receiving user).

- Fast start: VoIP applications are based on codecs which usually have a minimal required transmit rate. However, starting the transmission of a stream with a somehow large byte rate does not fulfill the principles of congestion control. Applications will probably have to cope with an initial slow-start period.

- Silence suppression: Most modern codecs use silence detection mechanisms and stop sending data packets when silence is detected, saving according to measurements up to 70% of the available bandwidth. While this change is quite steep for the codec generated data, when the codec starts transmitting again the bandwidth increase will be again gradual

(see the previously discussed fast start problem). The alternatives discussed on the DCCP mailing list are: sending packets of varying size at a constant rate with padding as content and using artificially generated noise for forcing the codec to generate data continuously.

- Bandwidth variation: This problem, while similar to the one discussed just before, is more likely to appear in combination with video codecs generated streams, and the problems that it poses are harder to solve than in the previous case.

  A video codec can generate compressed frame data by calculating the differences between successive frames. While between some frames these differences are small, a change of the scenery for example can require a so-called synchronization frame, generating additional data. According to [22] these rate changes can have a relative factor of ten for example, and can occur from one packet sent to another one, leaving a rate-controlling strategy practically no time to adjust. Buffering-averaging methods could be used for smoothing this fast increase, however they might contribute to a depreciation of the quality of service perceived by the user. While the factor of change for pure VoIP applications is such that the current rate control strategy could adapt to it, one should consider the advance of voice transmissions when designing a comprehensive strategy.

RFC 3448 [17] defines the TCP-Friendly Rate Control (TFRC) an alternative to the TCP rate control method defined in RFC 2581 [11]. While TCP rate control uses the Additive Increase Multiplicative Decrease (AIMD) method, TFRC has a smooth response to packet loss.

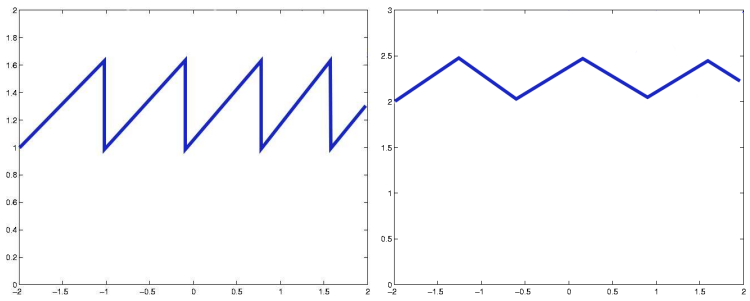

Figure 3.1: TCP vs. TFRC rate control

It can be assumed, for an increase of generality that the application can use a few nominal connection rates for sending data, resulting in different perceived quality. This assumption gives more flexibility in constructing an appropriate strategy.

In order to deal with the problems described above, the following strategy has been proposed (Strategy 3 in the original document [22]):

- Startup rate increase: The connection setup should include an initial rate increase phase in which no actual voice communication takes place. After

the initial connection establishment (for example through SIP messaging) the two parties exchange idle data in order to achieve an appropriate rate for beginning voice transmission. If this rate is not achieved within a reasonable amount of time, the connection is interrupted and the call is cleared.

- Congestion compensation: If congestion occurs during a call, TFRC will reduce the sending rate, and the application can switch to a lower rate codec. However, the application should pad its transmission to the allowed TFRC rate, in order to determine TFRC to increase the sending rate such as to allow the return to a higher-rate codec. This padding is only necessary if the rate increase is desired.
- Playout buffer: A playout buffer of about 100 ms can reduce the jitter experienced by the end-user.

The TFRC-VoIP mode comes to address further issues. This special TFRC mode addresses applications using small-sized, frequently transmitted frames in order to improve interactivity. According to the original document [22], usual voice packet data sizes are 80 to 320 bytes, well-under the 1480 bytes MTU of typical IP packets.

This can lead to problems, since while some network limitations are in bytes per second, others (the ones originating in routers for example) might be in packets per second, giving a DCCP stream less bandwidth than the one allocated to a comparable TCP stream. The VoIP mode of TFRC is designed to address such difficulties in applications using a transmission interval of at least 10 msec between packets.

3.2 Factors affecting VoIP transmission quality

A typical VoIP application takes sampled and digitized equal-length intervals of an audio signal which it compresses using a specialized codec, packetizes it and sends it over the network. The receiver de-packetizes the data and places it in a playout buffer, which has the role of compensating for the jitter that occurs during transmission. The data is then decompressed and the voice signal is replayed.

The sensible points influencing the perceived quality of the output signal are the codec performance, depending also on the bandwidth usage, the amount of networks loss, which can be to a point compensated by packet loss concealment methods, the network delay as well as the overall measured delay and the network jitter (the variance of the network delay). From the factors mentioned above, the network loss, network delay and jitter pertain to the underlaying network and can be traced back to congestion situations, queuing strategies and congestion control strategies while the other factors are application specific. Jitter, for example, depends heavily on the intermediate router's queuing behavior and the presence of alternative routes between the endpoints.

On the application side, VoIP applications use adaptive playout buffers in order to compensate for the jitter effect. Usually periods of silence are used for adapting the buffer's size. Moon, Kurose and Towsley present strategies that can be used by applications in order to compensate for network jitter using adaptive playout buffers, and give bounds on their possible performance [23].

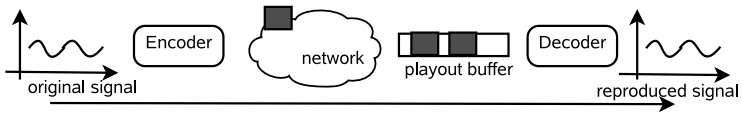

Figure 3.2: VoIP transmission sequence

3.3 Perceived quality models

The interest in the assessment of end-to-end perceived quality of voice streams has first appeared in the research of the telephony companies. Algorithms for the qualitative evaluation of the transmitted signals relay on psychoacoustic models, and due to their high complexity usually are applied on offline data samples which are compared to the original ones.

Generally the measurement unit for the perceptual quality of a voice transmission is the Mean Opinion Score (MOS), a subjective quality score ranging from 1 (unacceptable) to 5 (excellent). These scores can be computed by using subjective test or by analyzing the system using an objective model. For example, the PESQ algorithm generates quality scores which correlate well with subjective tests. The E-Model provides another way for generating quality scores and is particularly suited to assessing the distortions which appear due to packet transmission processes.

3.3.1 PESQ

The International Telecommunication Union (ITU) has developed the PESQ algorithm as Recommendation P.862. PESQ is able to predict perceived quality in a very wide range of conditions, including coding distortions, errors, noise, filtering, delay and jitter. The techniques used by the PESQ algorithm are documented in a series of articles by its authors. [24] [25]

The first problem that PESQ tries to solve is the detection of constant delay and jitter in the output sample. The algorithm does not use classical transfer function estimation or windowed cross-correlation techniques, but introduces a histogram-based technique.

The reference and measured signals are filtered through a high-pass filter during pre-processing. The motivation of this action is that while most of the typical speaker's voice signal's energy is concentrated in the low-band (under 500 MHz) the components of the voice spectrum that influence the intelligibility of the signal lie in the high-band.

A first approximation of the delay is given by the maximal cross-correlation point of signal envelopes, 4-ms long nonoverlapping signal frames. Due to the fact that the signal is highly nonstationary at this time scale, this indication is likely to be quite precise. The approximated delay is afterwards eliminated from one of the signals through a time shift.

In what follows the signals are divided into 64-ms 75%-overlapping frames and the position of the maximal cross-correlation is computed for each frame. The frame delays are then weighted according to their loudness and a running

average of these is the first estimate of the actual delay. This running value is smoothed by convolution with a triangular filter, obtaining the delay estimate.

In packet-switched networks jitter is more common than constant delays. Therefore the above-mentioned technique has been extended for the detection of jitter by using different frame types. The underlying assumption of the algorithm in this situation is that in the audio data corresponding to a certain transmitted packet the jitter is constant.

The paper [26] describes the PESQ algorithm in more detail, with an emphasis on its quality computation technique. In PESQ the audio signals are transformed into a psychoacoustical model representation, which in this case is calculated on the basis of signal representations that use the psychophysical equivalents of frequency and loudness. The difference between the internal representations of the input and the delay-compensated output represents the audible difference of the two samples, which will in turn be used to predict the perceived quality degradation.

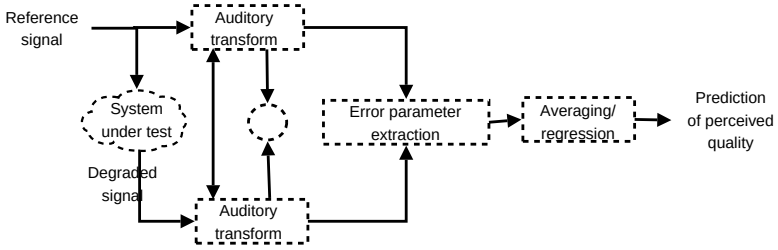

Figure 3.3: PESQ algorithm assessment cycle

The first step of the quality measurement is the calibration of the output signal for eliminating gain both in time domain and frequency domain by using test signals. The resulting signal filtered in order to simulate the effects of a headset on the output signal and its active speech part is isolated. In order to simulate the time-frequency decomposition performed by the human ear, a short-term FFT is applied to a window of 32-ms frames, overlapping in a proportion of 50%, obtaining the power spectra of the original and degraded signals. Effects that do not influence the psycho-acoustical perception, such as linear gain or short time-varying gain are compensated. Afterwards the so-called loudness density is computed, an indicator whose difference between the original and degraded signal is used for calculating the disturbance density, which shows the effects of noise on the signal.

The above-mentioned and other indicators have been correlated with the results of qualitative measurements performed by human subjects in order to train it for independent usage. The trained PESQ algorithm has a correlation rate of 0.935 with the human tests, making it a highly reliable quality measurement technique.

While the PESQ algorithm is particularly well-suited for assessing the distortions of the sound during transmission over a line which induces signal changes, it was not designed for the realm of digital, packet-switched networks. Its complexity makes it unsuitable for online quality comparison, and the final quality

score that it provides is dependent heavily on factors such as codec characteristics, and in order to infer the exact effect of the network state parameters on the quality score of the received signal further analysis will be required.

Given a time series representing packet end-to-end times or loss marks, we would like to be able to estimate the perceived quality of the corresponding voice transmission.

The preferred scale for classifying the perceived quality of a voice connection is the *mean opinion score* (MOS), used in subjective quality evaluation tests, and ranging on a scale from 1 (unacceptable) to 5 (best).

The International Telecommunication Union (ITU) has published the PESQ algorithm as Recommendation P.862. PESQ is able to predict perceived quality in a very wide range of conditions, including coding distortions, errors, noise, filtering, delay and variable delay (jitter). The techniques used by the PESQ algorithm are documented in a series of articles by its authors. [24] [25]

3.3.2 E-Model

The ITU-T E-Model [27] is an online analysis method generating a MOS score. The E-Model defines a quality factor R, from which he MOS score is obtained through the equation:

$$MOS = 1 + 0.035R + 7 * 10^{-6}R(R - 60)(100 - R)$$

illustrated in Figure 3.4

The authors have emulated the speech quality evaluation process of PESQ in measuring the quality degradation due to individual frame losses and have formulated a theoretical model for both distant and burst losses. In the case of burst losses correlation effects leading to increased perceptual distortion had to be accounted for.

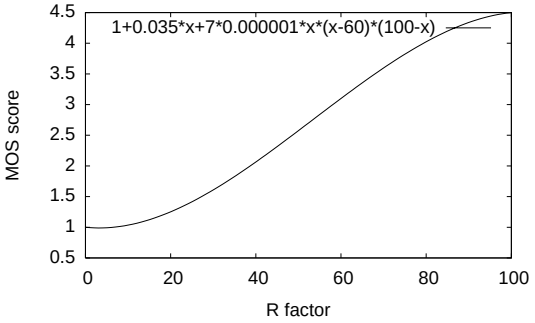

Figure 3.4: R factor - MOS score relationship

The E-Model has been further simplified [28] [29] for approximating the quality of VoIP-transported conversations. Their model approximations are capable, for the different codecs considered, to offer a quality statistic starting from the series of delays and losses associated with packet transmission. In the following chapter the E-Model based evaluation methods for packetized audio transmission will be presented in the form of a quality metric.

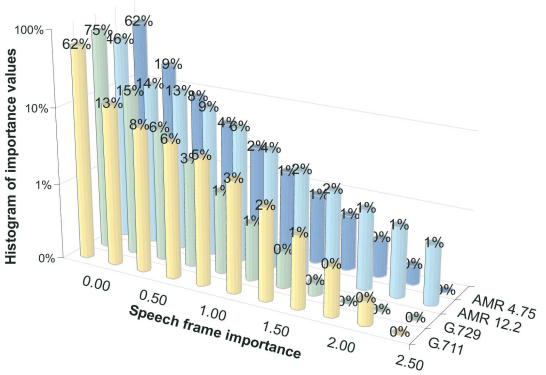

Figure 3.5: Distribution of the packet importances [30]

3.4 Statistical Properties of Voice Streams

Hoene, Wiethoelter and Wollisz [30] have used the PESQ analysis method in order to study the impact of different types of packet drop sequences upon the perceived speech quality. They compared the difference between the original audio streams and the streams decoded in the absence of individual frames. The result of their MOS measurements for an 8s long sample with a varying starting position and content of the dropped packet is shown in the figure 3.5:

The authors introduce a metric for the importance of the individual frame. Let $MOS(s)$ be the MOS score of sample with no packet drop, $MOS(s, e)$ be the MOS score under packet loss, with e being a vector describing the packet losses and $t(s)$ be the length of the respective sample. The importance $Imp(s, e)$ is defined as:

$$Imp(s, e) = (MOS(s) - MOS(s, e))t(s)$$

The above figure shows that the importance of most frames is relatively negligible.

The authors have emulated the speech quality evaluation process of PESQ in measuring the quality degradation due to individual frame losses and have formulated a theoretical model for both distant and burst losses. In the case of burst losses correlation effects leading to increased perceptual distortion had to be accounted for.

As online evaluation algorithms for evaluating the importance of different speech packets emerge, discriminative drop of packets might help increase the quality of the arriving signal. This idea is again approached in a following section.

3.5 Optimizing the Perceived Quality

As previously noted, streaming media packets have a largely varying impact on the quality of the re-combined signal. For example, statistical measurements show that few encoded speech packets have a high importance and standards

such as JPEG use synchronization frames which influence a longer series of packets.

The congestion prevention characteristics of DCCP make it possible to select before the actual transmission which packets should be dropped from the outgoing queue and allows the regulation of the queue's behavior through a series of informed decisions. While, upon congestion notification, DCCP could inform the application of the new data transmission rate and request that the outgoing data be replaced with data encoded at this rate, it seems far-fetched to assume that an application could react in all cases appropriately fast. However, assuming that the outgoing queue's de facto rate is larger than the current transmit rate, and that the application is interested in the timely transmission of data, it is likely that the kernel will have to decide which packets should be transmitted and which packets should be dropped.

We propose, therefore, an extension to the classic packet sending API, allowing an application to inform the kernel of the pre-computed importance factor of a given packet. This information could be sent either as an extra parameter of the send function, or as ancillary data associated with DCCP packets. The second information needed for applying this behavior is a time-stamp on the outgoing packets, allowing the kernel to decide on their expiration moment. The kernel might apply this timestamp during the execution of the write() call.

The kernel strategy would be to select the combination of packets for transmission that, by accumulating their individual importances, maximize the total importance and can be sent, at the momentary data rate, without exceeding the expiration time of any of the packets included. Given that each packet has an importance(gain) and a size factor(weight), this problem is an instance of the well-known backpack problem.

3.6 Adaptive Codecs

Speech compression codecs for voice produce a stream of equally time-spaced packets with their size varying depending on the amount of audio information to be transmitted. This behavior is implemented using the so-called variable bit rate (VBR) compression technique. Moreover, in periods when silence suppression is active the codec can completely stop encoding. Therefore, a certain variation of the bandwidth usage over time is to be expected, and since the application has an interactive nature, requiring timely delivery of packets, this variation must be permitted.

The same speech compression provide often an option for setting the target average bit rate (ABR), which the codec is supposed to target when performing VBR encoding. Applications can choose an encoding rate appropriate for the available bandwidth, adverting packet loss, or can decide to drop the connection if a minimal bandwidth is not available. The quality decrease due to a lower bandwidth should be more acceptable than the one due to packet drops.

The application should be able to detect the admitted transmission rate. Two methods are available:

- Reading through the socket API the data necessary for computing the instantaneous transmit rate. It appears that at this moment the API does not report the whole set of data needed for such an operation: while

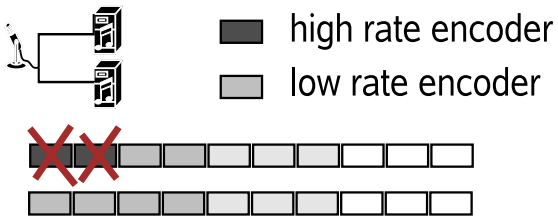

Figure 3.6: After congestion is detected, the high quality packets are replaced with smaller, lower quality packets which have a higher chance of arriving at the destination.

the transmission window size is available, the RTT measured value is also necessary.

- Measuring the delay induced by write() calls, and trying to approximate the transmit rate from it. This method is less precise since it involves user-space measurements, however, its accuracy might suffice for this purpose.

Based on this measurement, an application can instruct the codec to modify the ABR for future packets, or, since voice encoding is a from a computational perspective relatively inexpensive, decide to re-encode those packets still awaiting to be sent (if an API such as the previously described packet-buffers is used). Moreover, if the application experiences a sudden drop of the transmit rate, it can choose to remove from the queue those packets which will be least missed qualitatively in the voice signal's reconstruction.

The current chapter addressed some of the issues appearing when sending VoIP traffic over a congestion-aware protocol. It continued by giving an overview of the assessment methods of the perceived quality of voice transmission, also in the context of packet-switched networks and discussed some statistical properties of voice streams that might be used in trying to optimize the quality of the transmission. Strategies for dealing with congestion build-up in the case of variable rate codecs were presented.

Chapter 4

Evaluating VoIP Quality

The current chapter presents the experimental setup and the metrics involved in the quality evaluations. The chapter begins with a description of the conversation model used for generating speech patterns, followed by the presentation of the metric used in evaluating the way in which packet transmission reflects in voice quality. Next, it presents an algorithm for bounding the performance of playout buffers, adapted for fitting the quality estimation requirements. In the end it introduces the experimental setup used.

4.1 Generating Conversation-Like Data

Sriram and Whitt [31] introduce an on/off model for conversational patterns, in which periods of voice activity of variable length alternate with periods of silence. In the current approximation of the model we have generated, using exponential distributions, periods of voice activity of 1 second average duration, followed by periods of silence of 1.5 seconds average duration.

The input audio sequence is obtained by concatenating periods of continuous speech with periods of silence. The resulting audio is processed through the Speex audio codec for performing voice activity detection, and the resulting audio frames sequence is padded and packetized in order to emulate either the G729 or the G711 codec.

A segment of the resulting speech pattern is illustrated in Figure 4.1.

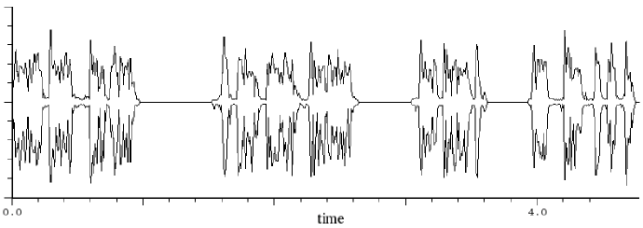

Figure 4.1: Resulting speech pattern

4.2 A Metric for Quality

The ITU-T E-Model [27] is an online analysis method generating a MOS score. The E-Model defines a quality factor R, ranging from 1 to 100 as:

$$R = R_0 - I_s - I_e - I_d + A$$

where the subfactors have the following meanings:

- R_0 - the effects of noises
- I_s - the effects of impairment occurring simultaneously with the voice signal
- I_e - the effects of impairment caused by losses
- I_d - the effects of impairment caused by delay
- A - compensates for the above impairments under certain conditions

The MOS score is obtained through the equation:

$$MOS = 1 + 0.035R + 7 * 10^{-6}R(R - 60)(100 - R)$$

In a VoIP system, due to the fact that delays and losses are the main affecting factors, the factors I_d and I_e are the only variables left. Cole and Rosenbluth have reduced the model to the following equation, using appropriately chosen default values for the canceled factors:

$$R = 94.2 - I_e - I_d$$

Their tests have shown that the following defining relation for I_e can be used in estimating the impact of loss:

$$I_e = \lambda_1 + \lambda_2 \ln(1 + \lambda_3 e)$$

where λ_1 quantifies the voice quality degradation due to the codec, λ_2 and λ_3 refer to the degradation due to loss and e represents the overall packet loss rate. Values for the λ parameters have been determined through simulations of different loss conditions for different codecs, and are reproduced in the Table 4.1 as published in [29] and [28].

Table 4.1: Coefficients for the calculation of the quality impairment due to loss [29] [28]

Codec	**frames / pkt**	λ_1	λ_2	λ_3
G.711	1	0	30	15
G.729	1	10	47.82	18
G.729A + VAD	2	11	30.00	16

Similarly the impact of delay has been modeled [28] as:

$$I_d = 0.024d + 0.11(d - 177.3)I(d - 177.3)$$

Table 4.2: Interpreting R factors and MOS scores[28]

R	MOS	Quality
90 - 100	4.34 - 4.5	Best
80 - 90	4.03 - 4.34	High
70 - 80	3.60 - 4.03	Medium
60 - 70	3.10 - 3.60	Low
50 - 60	2.58 - 3.10	Poor

where $I(x)$ is a unity step function and d is the total end-to-end delay. The form of this equation is justified by the experimental observation that after a threshold calculated at 177.3ms delays lead to a faster perceptual degradation.

The same authors interpret the MOS scores as indicated in Table 4.2.

The R factor under different loss and delay conditions is illustrated in Figure 4.2 for the G729 and G711 codecs.

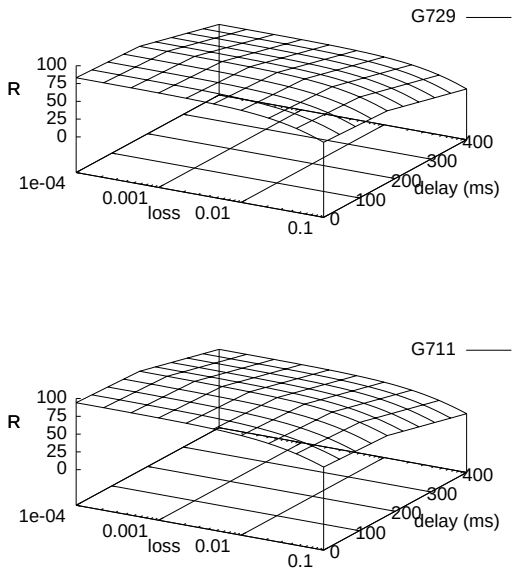

Figure 4.2: R factor variation under delay and loss conditions for the G729 and G711 codecs

4.3 Playout Buffer Performance Bounds

Cohen [32] was the first to mention the effect of jitter on audio packets traversing multiple networks.

Packets arriving at the receiver suffer different delays in the network. In order to provide a quality listening experience an audio application will try to replay them with a relative delay corresponding to their sampling period.

Arriving packets will be placed in a so-called playout buffer and delayed in order to achieve the same end-to-end delay for almost all/all packets within a talkspurt. Packets that cannot be played within this bound will be silently dropped.

Figure 4.3 gives a graphical depiction of the playout-buffer's operation. For each talkspurt a buffer management algorithm fixes a certain target delay (the elevation of a line of slope 1 separating the played packets from the packets silently dropped). The figure suggests that adjusting this value brings along a compromise between end-to-end delay and the loss percentage of the connection.

Different online management algorithms have been researched, published and are in usage in current applications (see [33], [23]), and the field is undergoing further research. Some of these algorithms have specific assumptions about the distributions of packet delays, for example they might be designed in order to cope with bursts of packets resulting from a congestion-relieved router. Currently, no buffer management method can be considered a standard, and different methods offer quite different delay/loss exchange rates.

For the purpose of evaluating the quality of packetized audio connections, some authors have proposed applying offline bounds on the performance of playout buffer management techniques. Rosenbluth and Cole [28] fix the end-to-end delay for the entire length of the conversation, and obtain their delay value by applying a Chernov bound on the arrival distribution. Tao and Guerin [34] define their fixed delay value as $arg\_max(R(d))$, where $R(d)$ is the quality function described in the previous section, applied on the series of packets resulting after selecting d as a maximal end-to-end delay threshold.

Assuming that the delays induced by the transport layer's congestion control mechanism might be unevenly distributed, using the same target delay for the entire conversation becomes inappropriate. A tight bound on the performance requires that every talkspurt, considered a minimal unit for which all packets must have the same end-to-end delay, be assigned its individually computed delay.

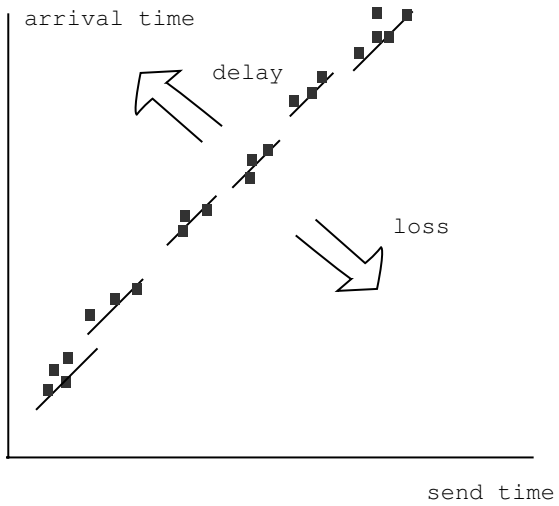

Figure 4.3: Playout buffer operation

Moon, Kurose and Towsley have introduced [23] an algorithm based on dynamic programming for computing the minimal delay of each talkspurt, assuming a maximum number of admissible losses caused by the output buffer in the whole conversational pattern. In the following we introduce their notation and give an overview of their method:

Let a trace consist of N packets grouped in M talkspurts. Define:

- n_k the total number of packets in the k-th talkspurt
- t_k^i, sender timestamp of the i-th packet in the k-th talkspurt.
- a_k^i, receiver timestamp of the i-th packet in the k-th talkspurt.
- d_k^i the normalized delay of the $i-th$ packet in the k-th talkspurt; the normalized delay is the packet network delay $a_k^i - t_k^i$ from which the minimal packet network delay in the talkspurt has been deducted
- $d_k^{(i)}$ is the i-th smallest normalized delay in the k-th talkspurt
- $D(k,i)$ the minimum average delay possible when choosing i packets to be played from the k-th to M-th talkspurt

Assuming that the packets in the different talkspurts cannot collide when played out as a result of delays, the authors derive the following equation:

$$D(k,i) = \begin{cases} 0 \text{ if } i = 0 \\ d_k^{(i)} \text{ if } k = M \text{ and } i \leq n_M \\ \infty \text{ if } k = M \text{ and } i > n_M \\ \min_{0 \leq j \leq i} \left(((i-j)D(k+1, i-j) + jd_k^{(j)})/i \right), \\ \text{otherwise} \end{cases}$$

The values $D(0,i)$, the minimal average delays of the entire connection, corresponding to different packet loss rates in the playout buffer, can be computed in $O(M \cdot N^2)$ time.

Knowing that each delay d has a corresponding impairment I_d, the method presented above can be modified in order to obtain a minimal averaged delay impairment (minimizing the value $D(0,i)$ does not automatically minimize the corresponding averaged I_d, due to the non-linearity of the delay impairment function). Let $I_d(k,i)$ be the minimum average delay impairment possible when choosing i packets to be played from the k-th to M-th talkspurt.

$$I_d(k,i) = \begin{cases} 0 \text{ if } i = 0 \\ I_{d_k^{(i)}} \text{if} k = M \text{ and } i \leq n_M \\ \infty \text{ if } k = M \text{ and } i > n_M \\ \min_{0 \leq j \leq i} \left(((i-j)I_d(k+1, i-j) + jI_{d_k^{(j)}})/i \right), \\ \text{otherwise} \end{cases}$$

For each i, after taking into account the network and sender losses, a corresponding loss impairment $I_l(i)$ is obtained. Denoting: $R(i) = 94.2 - I_d(0,i) - I_l(i)$, choose $R = R(i)$, where $i = arg\_max_i R(i)$ to be the upper bound on the audio quality of the trace.

Note: In the previously presented algorithm for estimating the best playout buffer decision it is assumed that all packets have a similar impact on the perceived quality. However, this might not be the case and algorithms based on more advanced quality measures might take a better-informed decision. For the E-Model based metric, the choice of dropped packets is not important.

4.4 Experimental Setup

The network setup is illustrated in Figure 4.4. It consists of three source machines, one router machine and one machine acting as a sink. The three sources are switched together to the same interface of the router;

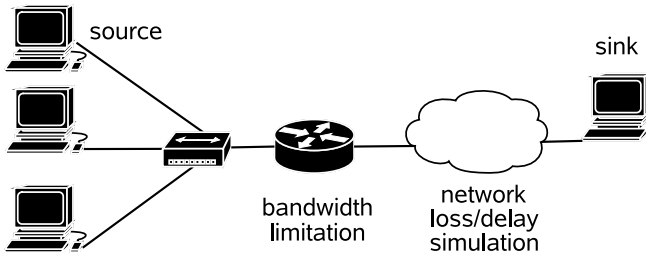

Figure 4.4: Network topology

Each one of the three sources is running four encoder processes, simulating the start point of an uni-directional voice connection (a DCCP half-connection).

Different network conditions are simulated by applying dummynet rules [35] on the router's outgoing interface. The experiments vary either the network delay between 0 and 400 ms, or the network loss ratio between 0.001% and 0.1%, present in both directions of the connection. The losses are uniformly distributed over time. We did not employ any models of Internet-specific or congestion-specific loss patterns, for example burst losses, since TFRC's loss interval approximation algorithms should be robust to such patterns.

The incoming interface of the router uses a 50 packet-long FIFO drop-tail buffer; for the multiple-connection experiments the buffer is bandwidth limited in order to cause apparent congestion.

The sender and the receiver track all codec frames. The sender logs all frames considered for sending before the send system call. The receiver logs all incoming frames after the completion of the recv system call. Further logging is performed by the router on the incoming and outgoing interfaces, using libpcap.

The current chapter presented the experimental setup and the metrics involved in the quality evaluations. The chapter gave a description of the conversation model used for generating speech patterns, followed by the presentation of the metric used in evaluating the way in which packet transmission reflects in voice quality. It presented an algorithm for bounding the performance of playout buffers, adapted for fitting the quality estimation requirements. In the end it introduced the experimental setup used.

Chapter 5

Experimental Results

This chapter presents the experimental results obtained and gives a discussion of DCCP's behavior in comparison to UDP and TCP.

5.1 Results

In contrast to TCP applications, UDP applications do not use a socket buffer in order to regulate the flow of data sent through the network, as packets written tend to be sent out immediately. For interactive applications the usage of a send buffer is unusual; however, for congestion control capable protocols, the application might prefer delaying the packets for small amounts of time instead of dropping them due to the momentary impossibility to send. Especially for applications that vary the rate of the data transmitted through the network by adjusting the packet size or stopping transmission frequently, such provisions might help in avoiding unnecessary packet loss during the time needed for probing for available bandwidth.

The KAME implementation has been modified during the project in order to allow the usage of a send buffer of configurable size. For the current experiments, the buffer size was set to a maximum of five packets, causing a maximal delay of 100 ms in the case of a 20 ms sampling rate.

The different DCCP congestion control modes have been compared to UDP, TCP using an increased initial window as specified in RFC 3390 [36] and TCP using the smaller initial window specified in RFC 2581 [11]. For the TCP transmission the TCP_NODELAY socket option was used in order to disable Nagle's algorithm for concatenating small buffer messages [9].

The following notations are used in the quality plots:

- *UDP:* The UDP Protocol
- *TCP:* The TCP Protocol, with the modification described in RFC 3390 enabled
- *TFRC:* TCP friendly rate control
- *TFRC SP:* TFRC small packets variant
- *TFRC SP FR:* TFRC small packets variant, with faster restart

- *TFRC SP FR MD:* TFRC small packets variant, with faster restart, modified in order to never decrease the sending rate below eight small packets per RTT; the minimal sending rate is computed as $\frac{8(s+H)}{R}$ where s is the average packet size, H is the packet header size, defaulted according to [18] to 40 bytes, and R is the RTT estimate.

Varying Delay

The first series of experiments varied the one-way delay of the connection between 25 and 400 msec. Due to the absence of losses, in this series of experiments, TFRC works only in the slowstart mode. For this reason, the faster restart variant of TFRC for small packets does not show a more aggressive response function than TFRC for small packets , since both variants specify that the transmission rate can at most double during one RTT in the slow start period.

The sending rate after periods of idleness was found to suffer due to a desynchronization between the idleness detection mechanism and the feedback mechanism, which reports the received rate X_recv. The reasons leading to this effect are discussed in the Section 5.2. We note that the TFRC SP FR MD version was not allowed to decrease the transmission rate under eight packets per RTT, despite the reported X_recv rate.

The effects can be observed in the Figures 5.1 a), 5.1 b) and 5.1 c), which show a sharp decrease of the quality score once the delay increases. The decrease is due to the large number of RTTs necessary for reaching the nominal send rate.

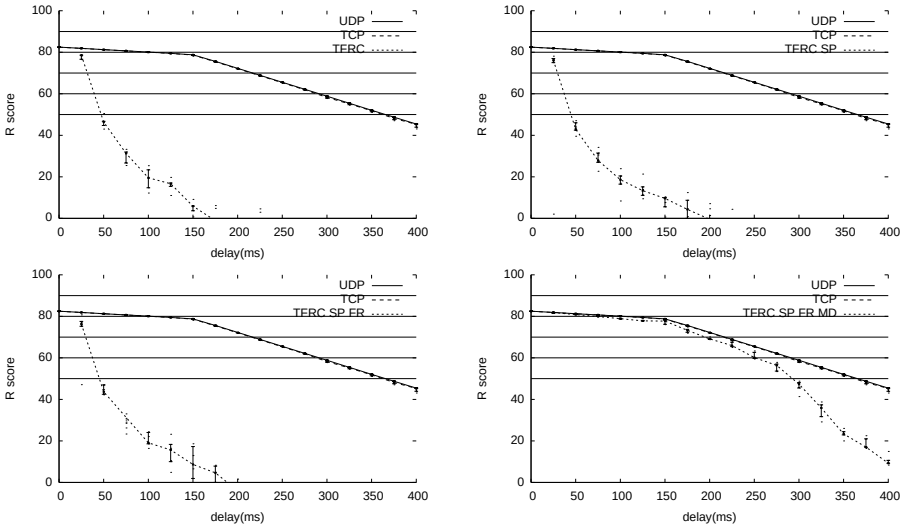

Figure 5.1: Quality plot - G729 - TFRC, TFRC SP, TFRC SP FR and TFRC SP FR MD under varying delay, median and quartiles illustrated

The only factor affecting the transmission is the above mentioned desynchronization between the idleness detection mechanism and the feedback mechanism causing a very low initial transfer rate after a period of feedback.

For TFRC for Small Packets, the limitation of the initial sending rate X_recv prevents the protocol from taking advantage of the change in the calculated

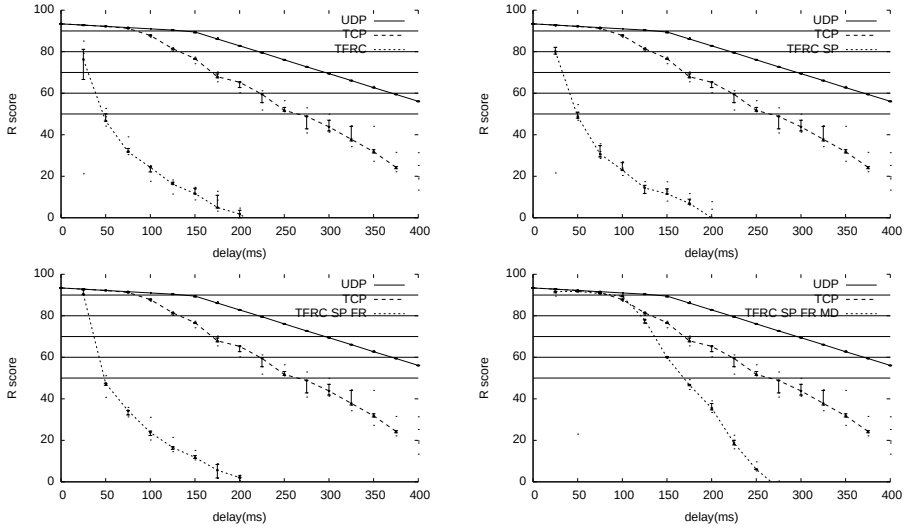

Figure 5.2: Quality plot - G711 - TFRC, TFRC SP, TFRC SP FR and TFRC SP FR MD under varying delay, median and quartiles illustrated

transmission rate. The achieved performance in this case is therefore similar to the one of TFRC.

Since the TFRC protocol under varying delay and no loss functions only in slowstart mode, the rate control of the faster restart variant of TFRC SP (Figure 5.1 c)) is identical in this case to the one of TFRC SP.

By allowing a transfer rate of eight packets per RTT at all times, TFRC SP FR MD (Figure 5.1 d)) allows a more free traffic dynamic, resulting in a less pronounced quality loss. Due to the low bandwidth of the codec, and to the fact that TCP uses a large initial window, the achieved performance is still inferior to the one of TCP. A quantitative analysis of the causes is given in the experiment discussion.

In the case of a higher bandwidth (Figure 5.3) the performance of all TFRC versions stays below the one of TCP, and is similar to the one in the previously analyzed case. The graphs illustrate that in this case, due to the fact that the initial congestion window is below the codec's nominal rate, TCP's performance is below the one of UDP.

Varying Delay and the Congestion Avoidance Mode

The following series of experiments revolves around the effects of TFRC's congestion avoidance mode on connections experiencing different delay conditions. In order to determine the switch from slowstart to congestion avoidance, a loss rate of 1/1000 packets is employed on the line. The loss rate has minimal effects on the perceived quality, but ensures that TFRC will function in the congestion avoidance mode for most of the connection time.

The effects can be observed in the Figures 5.3 a), 5.3 b), 5.3 c) and 5.3 d).

The regular TFRC variant (Figure 5.1 a)), described in RFC 3448 [17] shows a strong impact of rate control upon the VoIP traffic dynamic, causing a sharp decrease due to higher delay.

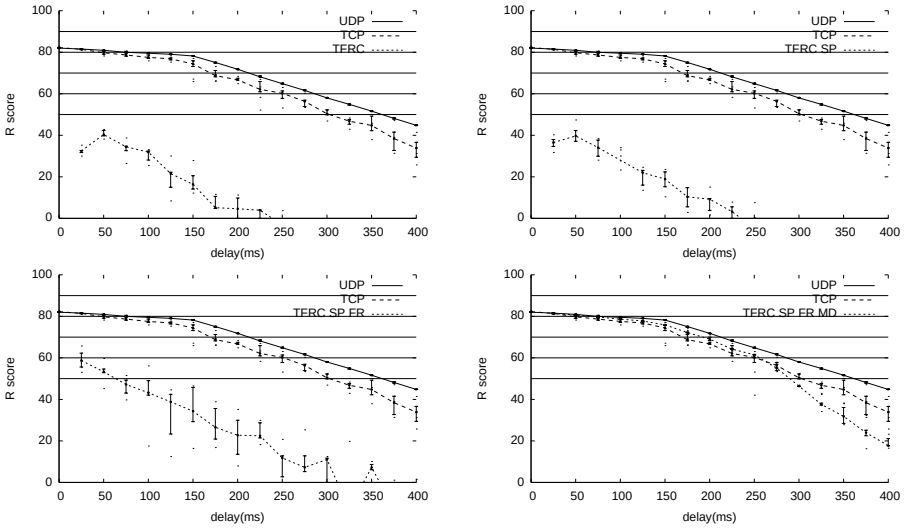

Figure 5.3: Quality plot - G729 - TFRC, TFRC SP, TFRC SP FR and TFRC SP FR MD under varying delay and small loss, median and quartiles illustrated

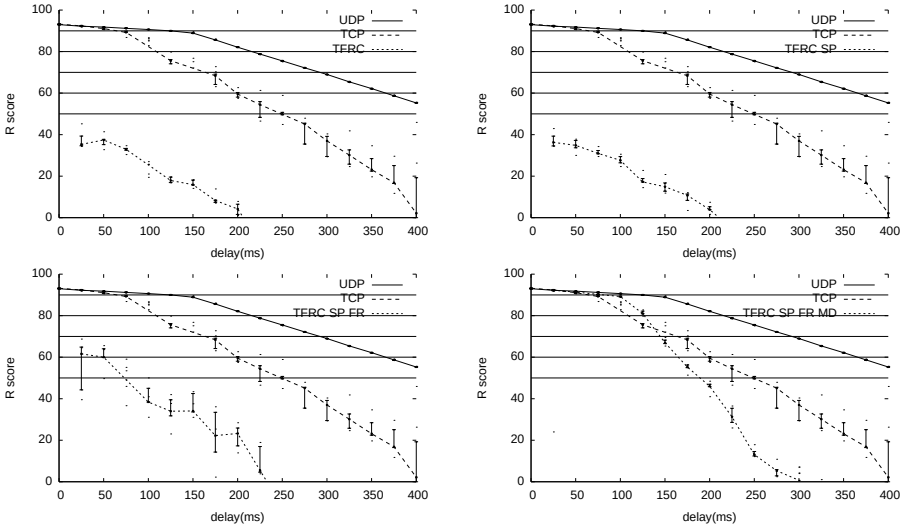

Figure 5.4: Quality plot - G711 - TFRC, TFRC SP, TFRC SP FR and TFRC SP FR MD under varying delay and small loss, median and quartiles illustrated

It is clear that plain TFRC is incapable of reaching the desired level of quality. The reason is that the rate equation of TFRC uses the actual packet size of the voice packets transmitted in deciding the maximal rate X_calc achievable by the protocol, while the TCP equation model on which the TFRC rate equation is based assumes that the packet size is equal to the maximum segment size. The resulting rate proves to be too constraining for allowing the transmission of VoIP traffic.

In the congestion avoidance mode TFRC's rate control (Figure 5.3 a)) is too constraining for allowing proper transmission. The cause is the low initial rate after periods of idleness, resulting from the reported X_recv value.

TFRC for Small Packets (Figure 5.1 b)) increases the equation computed sending rate X_calc by allowing the same transmit rate as the one of a TCP stream using 1500 bytes-long segments.

In the same mode, the TFRC SP variant (Figure 5.3 b)) suffers equally due to the low reported receive rate. The more permissive X_calc rate seems to have no impact on the quality of transmission, although it is higher than the codec's nominal transfer rate. The main factor leading to the quality degradation is the high number of RTTs required to reach the nominal rate after idleness, causing both delays and losses.

The faster restart variant of TFRC SP (Figure 5.3 c)) now has the possibility of quadrupling its transmission rate during one RTT, and thereby achieving superior performance. The performance gain is clearly visible when comparing to the slowstart mode variant (Figure 5.1). However, this advantage is only present in the congestion avoidance mode, and the transition from slowstart to congestion avoidance only occurs after the first loss event. In order for the voice traffic to be able to take advantage of the more liberal rate control offered by the faster restart version in all conditions (including the absence of losses), it would seem appropriate to modify the rate control algorithm for allowing the quadrupling of the transmission rate in the slowstart mode.

TFRC SP FR MD (Figure 5.3 d)) achieves a performance similar to the one of TCP. It can be inferred by comparing the quality impact illustrated in figures 5.3 d) and 5.1 d) that by allowing the quadrupling of the transfer rate also in the slowstart period (constituting the beginning of the conversation), the modified variant would be able to achieve an even better performance.

In the case of a higher bandwidth (Figure 5.3) the performance of all TFRC versions stays again below the one of TCP. Due to the faster rise of the transmission rate after talkspurt start, TFRC SP FR MD achieves a performance closer to the one of TCP.

Varying Loss

The loss experiments vary the loss factor on a logarithmical scale from 1/10000 up to 1/100 packets and a constant 50ms delay.

The interesting point of these experiments is the way in which DCCP's non-reliable transmission compares to TCP for which retransmission of lost packets is necessary. The second behavior to be observed is the way in which DCCP's rate adjustment due to measured loss affects VoIP transmission.

TFRC's rate (Figure 5.5 a)) is mainly affected by the reported receive rate X_recv. The final drop in quality occurs due to the computed rate X_calc.

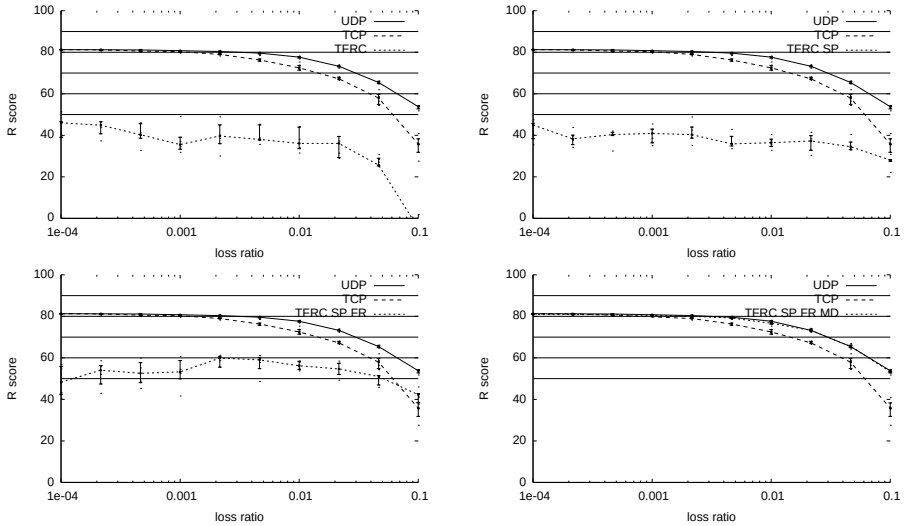

Figure 5.5: Quality plot - G729 - TFRC, TFRC SP, TFRC SP FR and TFRC SP FR MD under varying loss and 50ms delay, median and quartiles illustrated

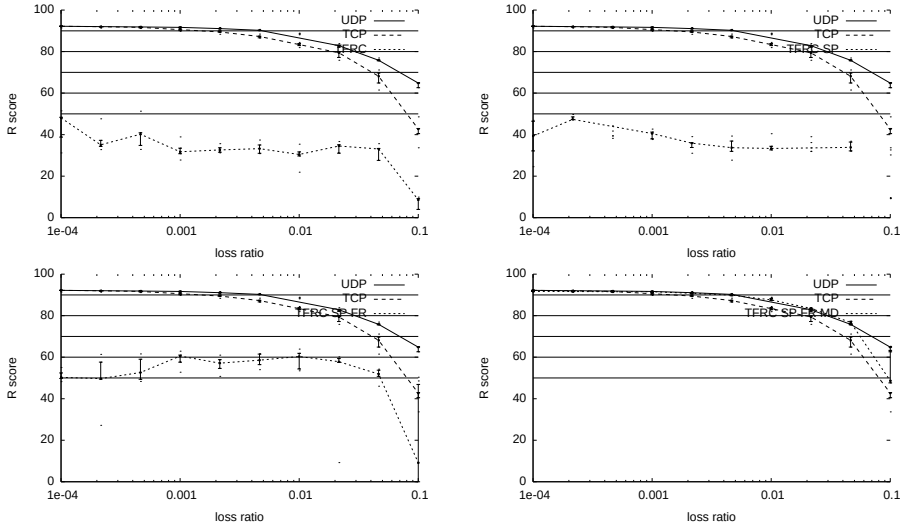

Figure 5.6: Quality plot - G711 - TFRC, TFRC SP, TFRC SP FR and TFRC SP FR MD under varying loss and 50ms delay, median and quartiles illustrated

TFRC SP's rate (Figure 5.5 b)) shows a similar behavior. However, the computed rate X_calc remains high even at higher loss rates and does not become a limiting factor.

In the case of the TFRC SP FR variants (Figures 5.5 c) and 5.5 d)) it can be noticed that the higher loss rate, by triggering an earlier transition from the slowstart mode to the less restrictive congestion avoidance mode, has a positive effect on the transmission quality.

As higher loss rates occur, TCP quality decreases due to the higher average delay incurred through packet retransmission. Although our experiments use a TCP SACK-enabled version, if the playout buffer decides to hold the retransmitted packet despite the occurring RTT delay, the whole sequence of packets in the same talkspurt will be delayed. Further complications might appear due to the arbitrary segmentation of voice data frames in the stream socket.

At a delay of 50ms, TFRC SP FR MD (Figure 5.5 d)) achieves a performance very similar to the one of UDP, and proves DCCP's advantage over TCP's guaranteed transmission schema in the case of interactive multimedia traffic.

In the case of a higher bandwidth (Figure 5.3) the effect observed are very similar. TFRC SP FR MD manages to overcome TCP and achieve a performance similar to the one of UDP, partly due to the absence of a retransmission mechanism.

In order to gain a better understanding of the effects of the rate limitation through the reported X_recv rate, several talkspurt periods are plotted in the Figure 5.7 for the TFRC SP FR variant.

The first subgraph shows the speech on/off pattern which serves as an input to the codec. The second subgraph presents a trace of the packets sent, of the packets received and for the packets played out it indicates the play out time. The third subgraph plots the send rate, recorded by the sender machine upon each run of the rate adjustment algorithm. The fourth subgraph presents the occupancy level of the sender buffer (which varies from one to five packets), recorded also by the sender machine upon each call of the dccp_output() function in the DCCP stack. The fifth subgraph plots the sizes of the transmitted packets at the input and the output interfaces, as extracted from the tcpdump traces of the router's incoming and outgoing interfaces. The last subgraph presents the quality impairments due to delay and to loss, computed over short time periods. The delay impairment is individual to each packet (while all packets in the same talkspurt are played with the same end-to-end delay) while the loss impairment is determined by computing a running average over 100 packets.

Figure 5.8 shows the same measurements in the case of the TFRC SP FR MD variant. The congestion effects present before have decreased, bringing a significant quality improvement.

Multiple Connections

The multiple connection experiments try to evaluate the relative fairness of simultaneously competing voice flows sharing a bottlenecked line.

The current chapter presented the experimental results obtained and gives a discussion of DCCP's behavior in comparison to UDP and TCP.

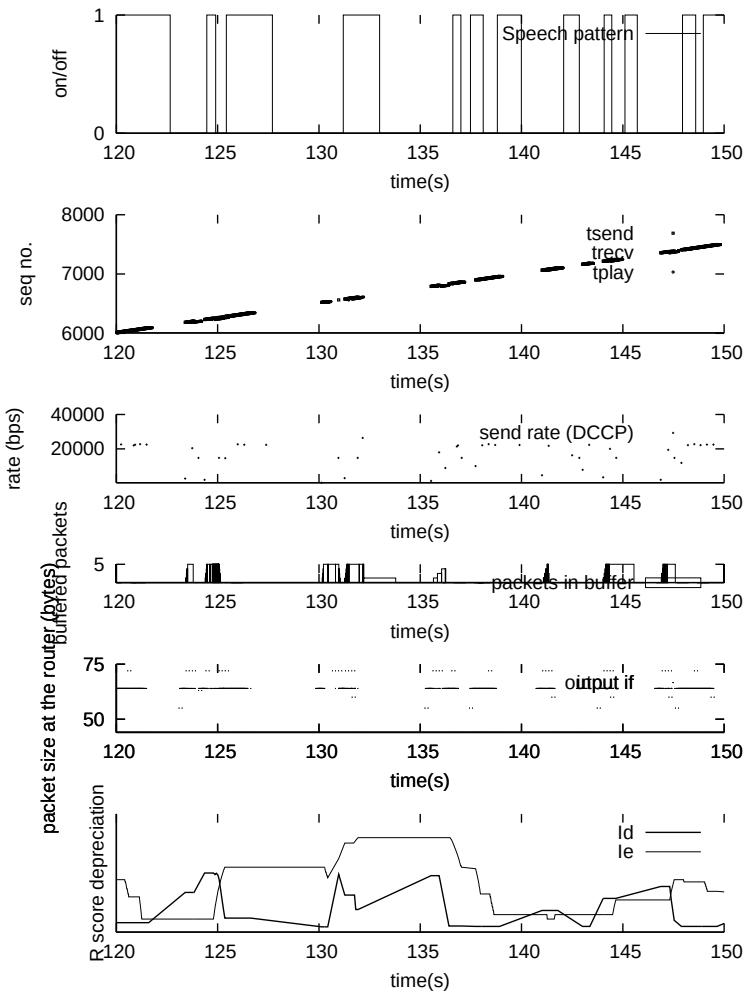

Figure 5.7: Single connection - G729, TFRC SP FR, send buffer of 5 packets, loss rate 1/1000, delay 150 ms

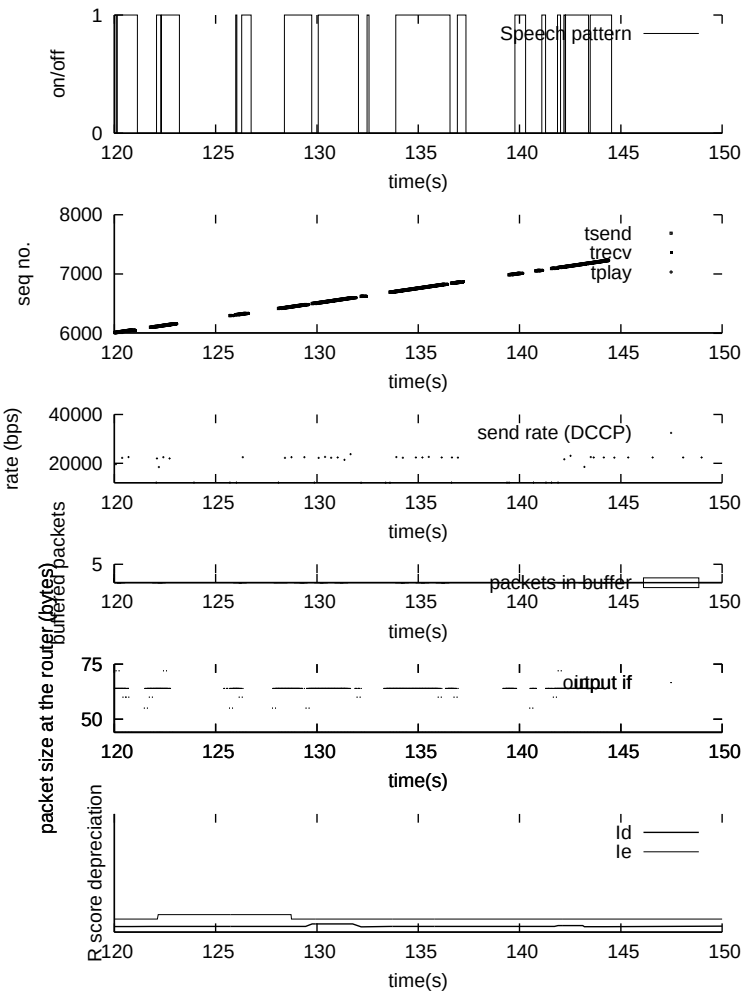

Figure 5.8: Single connection - G729, TFRC SP FR MD, send buffer of 5 packets, loss rate 1/1000, delay 150 ms

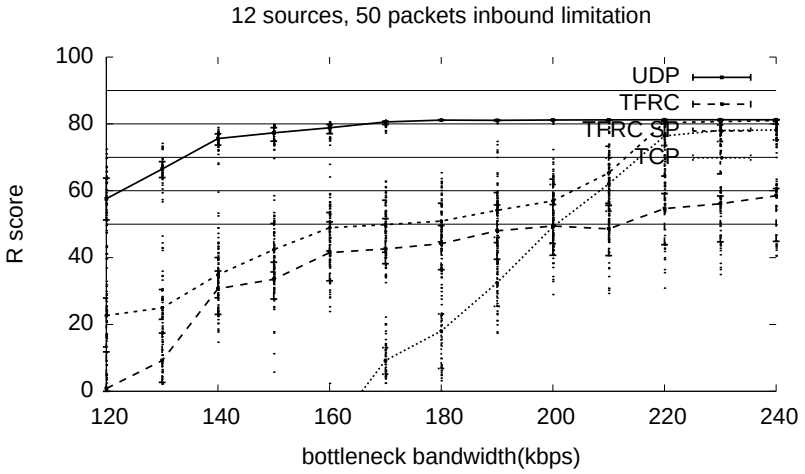

Figure 5.9: Multiple connections, G729

5.2 Discussion

The experiments presented in the previous section have evaluated the performance of the various variants of the TFRC protocol in transporting voice traffic, comparing it to the performance achieved by UDP and TCP.

The performance of all variants of TFRC was found to suffer because of a desynchronization between the idleness detection mechanism and the feedback mechanism. In order to estimate the actual impact that the small receive rate reported has on the quality of transmission, a different variant of TFRC for small packets with faster restart, that never allows the transmission rate to drop below eight packets per RTT, was tested.

In the slowstart period, the initial rates after periods of idleness are s/R for TFRC, TFRC SP and TFRC SP FR. However, this rate is practically not achieved due to the rate limitation brought by the reported receive rate X_Recv. More specifically, the feedback mechanism specified by CCID 3 [8] relies on window counters marking the data packets. The counters have values between 1 and 16, and the counter value of the sent packets is increased four times per RTT. The receiver will send a feedback packet whenever it receives a packet marked with a counter value exceeding by at least four the counter value of the last received packet, as measured in the cyclic number space of the window counters. Therefore, during standard operation (frequent arrival of data packets), the receiver sends a feedback packet about once per RTT.

Each feedback packet contains the reported value of X_recv, computed by dividing the number of bytes received since sending the previous feedback packet by the time elapsed since the previous feedback send event. After a period of idleness, one of the first data packets will determine the receiver to report a receive rate obtained by dividing the number of bytes corresponding to the payload of only a few packets, divided by the entire length of the idleness interval. Although RFC 3448 [17] specifies that the sending rate should never be decreased under four packets per RTT during a period of idleness, this provision

does not enter into effect because the feedback packet has been received as a response to a data packet, once the period of idleness has ceased.

The effects can be observed in the Figures 5.1 a), 5.1 b) and 5.1 c), which show a sharp decrease of the quality score once the delay increases. The decrease is due to the large number of RTTs necessary for reaching the nominal send rate.

RFC 2581 [11] specifies a minimal window for a TCP connection exiting an idleness period of:

$$IW = 2 * SMSS$$

while RFC 3390 [36] increases this value to

$$IW = min(4 * SMSS, max(2 * SMSS, 4380bytes))$$

where SMSS stands for the sender's maximum segment size.

Because the MSS values, either discovered by applying the path MTU algorithm or set by vendors at a "safe" value of 1500 bytes, are considerable larger than the size of typical voice packets, and taking into account that, opposed to TFRC SP and its variants, TCP does not take into account the packet header size in computing the transmission rate, it can be deduced that initial send rate of TCP is larger by around one order of magnitude than the one of TFRC SP FR MD.

Assuming a time interval of t_c seconds between the frames generated, a frame payload of s bytes, a packet header size H and an initial rate X_0, the TFRC with faster restart sender acting in congestion avoidance mode will need about $\log_4 \frac{(s+H)}{t_c \cdot X_0}$ RTTs in order to reach the nominal send rate, after a period of idleness. Depending on the buffer size, delays or losses will occur, causing the quality degradation seen in the graphs of the previous section.

The modified version of TFRC with faster restart uses a minimal sending rate of eight small packets every RTT. For comparison, RFC 3390 [36] specifies that a TCP connection allows an initial rate of at most four maximal segment sizes per RTT for a connection restarting after idleness. The initial rate in the case of TCP is larger by an order of magnitude than the one of TFRC. The experiments using the G729 codec clearly demonstrate the advantage that TCP has over the TFRC SP FR modified variant in this case.

The last possible source of problems found was the method used for the calculation of the length of the first loss interval while passing from the slowstart mode to the TFRC congestion avoidance mode. The computed loss interval is based by the rate observed during the last RTT before the loss event occurs. For VoIP traffic, presenting frequent periods of idleness, the rate observed might be significantly under the actual desired rate of transmission leading to a higher apparent loss rate and an artificial rate limitation.

Simultaneous DCCP traffic flows sharing a bottlenecked link were found to be relatively fair to each other. No starvation effects have been observed.

A possible solution for dealing with the slowstart behavior after idleness is to send data at a rate representing a fraction of the nominal codec rate during periods of silence. While this solution solves some of the shortcomings of the protocol, as demonstrated by a series of test performed, it is impractical in certain scenarios where the ratio of voice activity duration compared to voice inactivity is quite low (for example audio conferences between more than two participants). Moreover, due to the fact that the packet payload for VoIP pack-

ets is comparable to the header size, the actual bandwidth economy becomes minimal.

By comparing the results obtained for the G729 and G711 codecs it has been observed that DCCP scales better than TCP to codecs using higher bandwidths. This finding is valid as long as the packet payload does not exceed the maximum segment size, which is the typical case for voice applications, and is a consequence of the fact that TFRC SP's send rate during slowstart, computed in number of packets over a time interval, is independent of the packet payload. The second necessary condition is that the equation-derived rate X_calc, expressed in bytes, has to be above the codec's nominal rate.

Chapter 6

Conclusion

The current chapter summarizes the work presented and indicates some possible future research themes relating to the quality of VoIP transmission over DCCP.

The focus of the current work has been congestion control for interactive media. We considered the problem of testing the quality of VoIP traffic transported over the DCCP protocol, using some of the different proposed variants of congestion control methods. The evaluation framework developed can be used in estimating the quality implications of any transport-level congestion control schema proposed for the DCCP protocol. By applying the evaluation technique to the currently proposed congestion control mechanisms, we were capable of exploring some of their existing limitations. We tried to show the ways in which congestion control interacts with the dynamic of interactive traffic.

The experimental results have indicated a desynchronization between the idleness detection mechanism and the feedback mechanism which reports the received rate, affecting the restart after periods of idleness.

Experiments have also shown that TFRC's method of setting the initial window after connection restart affects the way in which voice connections restart, depending on the connection delay. We have compared the different TFRC variants with TCP in order to asses their impact on different codecs.

The last possible source of problems found was the method used for the calculation of the length of the first loss interval while passing from the slowstart mode to the TFRC congestion avoidance mode.

The effect of different delay and loss conditions in combination with the TFRC traffic congestion methods has been evaluated.

We believe that the current work brings forward some of the issues and constraints present in designing congestion control schemes for interactive media.

In the following a number of questions appearing during our investigations are presented. For gaining a complete understanding of the way in which data and voice flows can coexist in best-effort networks, further investigation of the following topics is required:

The possibility of using optimized send buffers for trying to improve the perceived voice quality has been mentioned already. To this point there are a number of possible implementation options, ranging from a modified send buffer

that drops the older packets in order to accommodate new ones, to buffers using packet priorities in order to make informed drop decisions and the packet-buffers described by Kohler and Padhye. The choice between mechanisms affects not only the application-level interface of the protocol, but also the complexity of the encoder-sender package. More advanced applications could use quality-assessment techniques in order to optimize their output for facing network congestion.

The paper [37] mentions that applications that fail to use their TFRC-allocated sending rate might encounter starvation when competing with TCP traffic in a FIFO queue. Since voice codecs have, in general, limited bandwidth usage, and do not aggressively probe for extra bandwidth, they are particular prone to suffer from this form of starvation.

The experiments have shown that adding one or two competing TCP streams to a pool of UDP or DCCP voice streams using a low bandwidth codec causes a drastic reduction in the voice quality.

There are different methods for dealing with the above-mentioned problem. Using adaptive codecs, an application can both offer the best quality allowed by the available bandwidth and prevent starvation due to a lack of aggressiveness in bandwidth usage. Applications might have to accept that competing TCP streams might cause packet loss at high levels, and use supplementary bandwidth for protecting their data payload through some form of redundancy such as FEC (forward error correction). Other strategies are based on queue management methods for routers which try to limit the delay of in-flight packets, such as RED. It is possible that these management strategies will have to be revised in order to be able to cope with the limited size of voice packets.

ECN can provide a dramatic improvement in the quality of multimedia transmission, by eliminating packet loss as a prerequisite of congestion recognition. Since voice stream quality is drastically affected even by small amounts of losses, and ECN can signal congestion build-up to both multimedia and data streams, sharing a link for both voice and data transfers while avoiding flow starvation and unnecessary delays becomes possible.

The last two methods mentioned require extensive deployment by service providers. The possibility of being able to mix data and multimedia traffic at the transport layer is a certain incentive for their widespread adoption.

Bibliography

[1] S. Floyd and J. Kempf, "IAB Concerns Regarding Congestion Control for Voice Traffic in the Internet," RFC 3714 (Informational), Mar. 2004.

[2] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," 2005.

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

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

[5] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion control without reliability," 2003.

[6] "Dccp state diagram image," 2006. [Online]. Available: http://portal.miun.se/ necu9500/dccp/start.html

[7] S. Floyd and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control," RFC 4341 (Proposed Standard), Mar. 2006.

[8] S. Floyd, E. Kohler, and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)," RFC 4342 (Proposed Standard), Mar. 2006.

[9] J. Nagle, "Congestion control in IP/TCP internetworks," RFC 896, Jan. 1984.

[10] V. Jacobson and M. J. Karels, "Congestion avoidance and control," *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, vol. 18, 4, pp. 314–329, 1988.

[11] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control ," RFC 2581 (Proposed Standard), Apr. 1999, updated by RFC 3390.

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

[13] W. Almesberger, J. Salim, A. Kuznetsov, and D. Knuth, "Differentiated services on linux," 1999.

[14] J. Padhye, "Model-based approach to TCP-friendly congestion control," Ph.D. dissertation, University of Massachussetts Amherst, 2000.

[15] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, 2000.

[16] P. Sarolahti and A. Kuznetsov, "Congestion control in linux tcp," 2002.

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

[18] S. Floyd and E. Kohler, "TCP Friendly Rate Control (TFRC): the Small-Packet(SP) Variant ," 2006.

[19] E. Kohler and S. Floyd, "Faster Restart for TCP Friendly Rate Control (TFRC)," 2005.

[20] N. E. M. Magnus Erixzon, Joacim Haggimark, "Implementing DCCP Design Specification," 2003.

[21] J. Lai and E. Kohler, "A Congestion-Controlled Unreliable Datagram API," 2004.

[22] T. Phelan, "Strategies for Streaming Media Applications Using TCP-Friendly Rate Control," 2005.

[23] S. B. Moon, J. Kurose, and D. Towsley, "Packet audio playout delay adjustment: performance bounds and algorithms," *Multimedia Syst.*, vol. 6, no. 1, pp. 17–28, 1998.

[24] Anthony W. Rix, Michael P.Hollier, Andries P. Hekstra, John G. Beerends, "Perceptual Evaluation of Speech Quality (PESQ) The New ITU Standard for End-to-End Speech Quality Assessment."

[25] A. W. Rix, M. P.Hollier, A. P. Hekstra, and J. G. Beerends, "Perceptual Evaluation of Speech Quality (PESQ) The New ITU Standard for End-to-End Speech Quality Assessment. Part I – Time-Delay Compensation."

[26] ——, "Perceptual Evaluation of Speech Quality (PESQ). The New ITU Standard for End-to-End Speech Quality Assessment Part II– Psychoacoustic Model."

[27] ITU, "Recommendation G.107. E-model, a computational model for use in transmission planning."

[28] R. G. Cole and J. H. Rosenbluth, "Voice over ip performance monitoring," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 2, pp. 9–24, 2001.

[29] L. Ding and R. A. Goubran, "Speech Quality Prediction in VoIP Using the Extended E-Model," *GLOBECOM IEEE Global Communications Conference*, vol. 7, pp. 3974–3978, 2003.

[30] A. W. Christian Hoene, Sven Wiethoelter, "Calculation of Speech Quality by Aggregating the Impacts of Individual Frame Losses," *Proc. of Thirteen International Workshop on Quality of Service (IWQoS 2005), Passau*, 2005.

[31] K. Sriram and W. Whitt, "Characterizing superposition arrival processes and the performance of multiplexers for voice and data," in *Proceedings of IEEE Global Telecommunications Conference, New Orleans, December 1985*, 1985.

[32] D. Cohen, "Issues in transnet packetized voice communication," in *SIGCOMM '77: Proceedings of the fifth symposium on Data communications*. New York, NY, USA: ACM Press, 1977, pp. 6.10–6.13.

[33] R. Ramjee, J. F. Kurose, D. F. Towsley, and H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks," in *INFOCOM (2)*, 1994, pp. 680–688.

[34] S. Tao, K. Xu, A. Estepa, T. Fei, L. Gao, R. Guerin, J. Kurose, D. Towsley, and Z. Zhang, "Improving voip quality through path switching," in *Proceedings of IEEE Infocom, 2005*, 2005.

[35] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.

[36] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC 3390 (Proposed Standard), Oct. 2002.

[37] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *SIGCOMM 2000*, Stockholm, Sweden, August 2000, pp. 43–56.