Implementation of the Deadline-based Selective Retransmission For Maintaining Low End-to-End Delays

Suhas Lande
Computer Science Department
University of North Dakota
Grand Forks, ND 58202-9015
suhaslande@yahoo.com

Jun Liu
Computer Science Department
University of North Dakota
Grand Forks, ND 58202-9015
jliu@cs.und.edu

Abstract

In real-time streaming media applications, data packets are typically valid for short time period. It is important to maintain a low and stable transmission latency in delivering packets for the real-time streaming media applications. Although both reliability and low latency are desirable for applications, maintaining a low and stable latency benefits more to applications than fully reliably delivering all packets. Selective retransmission of some packets help to achieve a tradeoff between reliability and latency. This paper presents the design and implementation of a deadline-based selective retransmission method. It allows a real-time streaming media application to reliably transmit the favorable packets by assigning longer deadlines to the selected packets. The transmission latency can be maintained low and stable when only a portion of all packets are reliably transmitted. Through actual network experiments, we demonstrate that a low and stable transmission latency can be maintained by adopting a deadline-based selective retransmission control method.

1 Introduction

Applications relying on real-time data streams have gained their importance over the past years. The correct functioning of such applications depends on the timely delivery of real-time data streams. Typically, each data packet in a real-time stream is only valid for a short time period, *i.e.*, each packet is usually associated with a deadline. When a real-time data stream has to be transmitted over a network from a data source to a data sink (*e.g.*, a processing center) where a stream processing application runs, both timely delivery and reliability are expected to be held in delivering the stream. Timely delivery of data packets makes packets to arrive at the processing center before their expiration. Reliably delivering data packets serves to make the applications running at the processing center side to correctly function. However, it is difficult to satisfy both timely delivery and reliability. On one hand, reliably delivering all packets makes it difficult to bound the time delay from the moment when a packet is available at the data source for transmission till the moment when the packet is delivered to the data sink. In order to reliably transmit every data packet, some data packets may have to be transmitted multiple times when the network becomes congested. Thus, a data packet may experience a long delay before its successful transmission. On the other hand, timely delivery of data packets make it difficult to reliably transmit all packets. In order to control the transmission delay, some packets may be given up their transmissions. Hence, appropriate transport protocols are needed to support the timely delivery of real-time data streams.

Continued transmission of a data packet beyond its deadline does not benefit the functioning of a real-time streaming application. Actually, transmitting expired data packets unnecessarily consumes network bandwidth and the processing power of a protocol stack. Real-time streaming applications typically are assumed to be able to prioritize the transmission of data packets. Hence, real-time streaming applications are benefited by allowing them to dynamically determine the transmission of data packets based on the deadlines of packets. Based on the status of transmission, a real-time streaming application can dynamically cancel the transmissions of packets, or update the content of a packet that has not been acknowledged.

The ability of allowing applications to dynamically determine the transmissions of packets is not readily available in existing transport protocols. Traditional transport protocols include the User Datagram Protocol (UDP) and the Transport Control Protocol (TCP). UDP is a connectionless and unreliable protocol, and TCP is a connection-oriented and reliable protocol. Each of the two protocols satisfies one of the contradicting requirements for delivering real-time data streams. UDP provides timely delivery of packets, but not reliability. On the contrary, TCP provides reliability, but not the timely delivery of packets. Over the recent years, the Datagram Congestion Control Protocol (DCCP) [1] has been proposed for controlling the time delay experienced by packets. DCCP is a connection-oriented transport protocol without the requirement of in-order delivery. DCCP enables the opportunity of achieving a compromise between timely delivery and reliability, so it is useful for applications with timing constraints. In the reliable in-order delivery, the packets have to be transmitted in order, and packets are kept retransmitted before they are successfully delivered. Without requiring the reliable in-order delivery makes it possible to maintain the timely delivery of some favorable packets by giving up retransmitting the packets that precede the favorable packets. DCCP has only specified the mechanism which detects possible packet losses, but it leaves the decision of handling packet losses to the applications which run on top of DCCP.

This paper presents the design and implementation of a deadline-based selective retransmission method. This method allows a real-time streaming media application to dynamically decide the retransmission of packets based upon the application's preference. It is assumed that every packet is associated with a deadline. An application can decide whether an expired packet should continue being retransmitted depending on the content of the packet. Flexible transmission arrangement can be enabled by assigning various deadlines to data packets of different importance. If a data packet is to be reliably transmitted, then it is granted with a relaxed deadline. A packet removal mechanism is developed to remove the expired packets from the list of pending data packets which are waiting to be transmitted. We demonstrate that a low end-to-end delay can be maintained by only reliably transmitting important packets in a flow.

The deadline-based selective retransmission method works at the user-level for facilitating user-level applications to control the retransmissions of packets. This selective retransmission method relies on the support of an underlying user-level transport functionality. This functionality is built based on a user-level implementation of DCCP, which is called DCCP/TP [2]. DCCP/TP is a fresh-start implementation of DCCP and is optimized for portability. DCCP/TP supports many DCCP features and implements reliable connection setup, tear down, and feature negotiation. DCCP/TP emulates the socket support which is traditionally implemented in the kernel of an operating system. DCCP/TP opens a regular UDP socket through which all DCCP packets are sent to/received from the kernel. DCCP/TP relies on the UDP/IP protocol stack that is implemented in the kernel. DCCP/TP generates and maintains sockets for DCCP connections. No matter how many DCCP connections are opened, the single UDP socket delivers all packets for the DCCP connections.

In our work, we have introduced the necessary components to the DCCP/TP, which allow an application to check the transmission status of packets and to express the instructions of transmission cancellation to DCCP/TP. Based upon the added supportive components, we implemented a deadline-based selective retransmission method. In this method, a data packet is kept transmitted till its successful delivery or its cancellation instructed by the application. An application can cancel the transmission of a packet based on various factors. In this paper, we only demonstrate the case when the cancellation decision is made purely on the expiration of a packet.

In the rest of this paper, the related work is presented in Section 2. A deadline-based selective retransmission supportive framework is described in Section 3. A deadline-based selective retransmission control method is described in Section 4. The ability of the deadline-based selective retransmission control method is described in Section 5. Our work is summarized in Section 6.

2 Related Work

The real-time transport protocol (RTP) [3] provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. RTP is an application-level protocol which defines a standardized packet format for delivering audio and video over the Internet. RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so. Namely, RTP only enables a real-time streaming application to form and control the packet streams at the application level. However, RTP can not match the transport properties with the intent of an application. One solution to address the need of matching is to use a configurable transport protocol (such as CTP [4]). A configurable transport protocol (in kernel space) supports the construction of customized protocols from collections of software modules, each of which implements a given transport property.

User-level networking infrastructure serves to improve network performance and to provide flexibility to applications, especially for the latency-sensitive streaming media applications. A number of user-level designs have been proposed in previous work.

- `icTCP` [5] is an "information and control" TCP implementation that exposes key pieces of internal TCP state and allows certain TCP variables to be set in a safe fashion. The primary benefit of `icTCP` is that it enables a variety of TCP extensions to be implemented at user-level while ensuring that extensions are TCP-friendly. By exposing information and safe control of the TCP congestion window, TCP protocols can be implemented at the user level.

- Alpine [6] is a user-level networking infrastructure for network protocol development. Alpine supports a FreeBSD networking stack on top of a Unix operating system. Alpine virtualizes the FreeBSD networking stack in user space without compromising the compatibility among kernel, networking stack, and applications.

- The Network Protocol Server (NPS) [7] is a real-time network system which is implemented on Real-Time Mach for supporting real-time communication. NPS provides a framework for implementing real-time network protocols which require to bound protocol processing time. The key feature of NPS is that the processing time of the network system is predictable because the priority of the network system is under good control in NPS.

- The U-Net communication architecture [8] provides parallel and distributed applications with a virtual view of a network interface to enable user-level access to high-speed communication devices. The U-Net architecture removes the kernel from the communication path and allows protocols to be constructed at the user level.

- An experimental user-level implementation of TCP [9] is TCP/IP stack with TCP in user space and IP in kernel space. Placing TCP in user space enables a flexible transport protocol which allows integrated communication subsystems.

- Jetstream Gbit/s LAN [10] is a set of user-space protocols which provide applications with a flexible low-level network interface to delivers delay-sensitive traffic.

> Jetstream frames contain a channel identifier so that the network driver can immediately associate an incoming frame with its application. Applications are enabled to delegate the processing of their data to the network interface without the need to first move the data into the application's address space.

Selective reliability provides an application with the flexibility of providing reliability to selected packets while constraining the transmission latency. The Image Transport Protocol (ITP) [11] is designed for image transmission over loss-prone congested or wireless networks. ITP improves user-perceived latency and achieves better interactive performance at the receiver. ITP runs over UDP and incorporates receiver-driven selective reliability to adapt to network congestion. ITP is a generic selectively reliable unicast transport protocol with congestion control that can be customized for specific applications and formats. Compared to the traditional but overly restrictive approach of transporting images using TCP, ITP allows a receiver application to improve the interactivity and responsiveness of reconstructed images.

3 The Basic Mechanisms Supporting The Deadline-based Selective Retransmission

We first describe the basic components of the user-space transport functionality. This functionality follows a client/server architecture. A stream source acts as a client, and the stream sink (*e.g.*, a processing center) acts as a server. A stream source feeds real-time data stream(s) to the stream sink where streams are processed. A stream sink can simultaneously accept multiple streams from multiple stream sources. The client/server structure is built based on the `Iperf` [12] package. DCCP is used as the transport-layer protocol supporting the communication between the client and the server . DCCP is suitable for applications that can benefit from control over the tradeoff between timeliness and reliability. Most of the control actions on a stream is performed on the stream source side. The transport functionality on the stream source side consists of three major components: the user-level transport protocol stack, the real-time stream generator, and the application-specific transmission controller.

3.1 The Client/Server Structure

The `Iperf` [12] package is used to establish the client/server structure. `Iperf`is a networking tool that is commonly used in measuring the bandwidth and the quality of a network link. In our work, `Iperf`is used to establish the application-layer protocol. The pair of `Iperf`client and server act as the application-layer end-points of a stream. The source of a stream is treated as the `Iperf`client, and the sink of the stream is treated as the `Iperf`server. The pair of `Iperf`client and server run on top of the DCCP protocol. The layered relationship between the `Iperf`client/server and the DCCP protocol can be likened to the layered relationship between the FTP client/server and the TCP protocol.

3.2 The User-Space Implementation of DCCP—DCCP/TP

The application-layer protocol (*i.e.*, the `Iperf`client/server) works together with a user-level implementation of the DCCP protocol. Traditionally, transport-layer protocols are implemented inside the kernel of operating systems. The existence of the barrier between the user space and the kernel space makes it difficult for a user-space application to control the kernel-space transmission of packets. Hence, we chose to adopt the user-space implementation of DCCP. DCCP/TP is a user-space implementation of DCCP, which is optimized for portability. DCCP/TP implements reliable connection setup, teardown, congestion control, and feature negotiation features of DCCP. DCCP/TP enables a user-space application to control the transmission of packets.

3.3 The Real-Time Stream Generator

Before transmitting a data stream, a stream source has to establish a connection with the designated stream sink. The connection establishment is performed by the underlying user-space DCCP/TP stack. Once a data stream finishes its transmission, the stream source closes the connection. There are two ways of generating an actual real-time data stream. First, a real-time data stream can be the a sequence of packets that are generated by a physical device. The dispersion in time between consecutive packets is determined by the time interval between the generation of the two packets. In this case, the stream generator is simply an entry point which accepts the packets fed by an external physical device and submits the packets to DCCP/TP for transmission. Second, a real-time data stream can be generated by an emulator which produces packets in real time according to a prescribed packet trace. In this case, the stream generator replays packets in real time. In this paper, we focus on describing the latter case because of the challenge of reproducing the time dispersion between packets. Generating a packet itself is not difficult, but the difficulty is to accurately reflect the actual time dispersion between packets.

- **The Real-Time Stream Source Emulator**
  The stream emulator simulates the arrivals of data segments and feeds the data segments to the DCCP/TP stack for transmission. Each emulator can only generate a single real-time data stream. A real-time clock unit is used to provide the timing support to emulators. The emulator reads the specifications of packets one at a time in the strict causal order. An emulator produces a packet based on the specification before making the next read. In order to produce packets at their prescribed timestamps, an emulator submits a timing request to the real-time clock unit and waits for the activation signal from the real-time clock unit. The real-time clock unit delays sending back an activation signal for a time period which is specified in the corresponding request. Once receiving an activation signal, an emulator produces a packet and submits the packet to DCCP/TP for transmission.

- **The Real-Time Clock (RTC) Unit**
  The RTC unit performs real-time timing for stream emulators. Since only one real-time clock is available in most of the low-cost real-time operating systems, a RTC unit is needed to service the timing requests for multiple stream emulators. Precise

timing may require special hardware support. Most of the low-cost real-time enabled operating systems often generate interrupt signals at constant frequency, *e.g.*, the real-time kernel patched from Linux [13]. In order to enhance portability, the RTC unit approximates a delay interval specified in a timing request by the closest time span of a number of consecutive interrupt signals.

The RTC unit maintains a linked list which holds the timing requests submitted by the stream emulators. At any time, each stream emulator can only have at most one timing request in the list. Each timing request is inserted into the list based on the delay period requested by a stream emulator. The head of the list contains the earliest expiring request among the requests that are currently in the list. The rest timing requests are sorted in the list according to their expiration times.

3.4 The Application-Specific Transmission Controller

The transmission controller enables a stream source to control the transmission of packets. A stream source is typically a component of an application which relies on real-time data streams. A data source can perform two types of controls through the controller: cancelling the transmission of an unacknowledged data packet, or updating the content of an unacknowledged data packet. When cancelling the continued transmission of some unacknowledged packets, the controller marks these data packets as *Cancelled* in the temporary buffer that holds the unacknowledged packets in DCCP/TP. The DCCP/TP stack then stops the continued transmission of the cancelled data packets and removes them from the temporary buffer. Meanwhile, an application can always update the content of an unacknowledged data packet before its successful delivery. The controller consists of two components: an interface and an application-specific transmission control method. The interface sits between the transmission controller and DCCP-TP, which allows an application to perform limited control on the transmission of packets. An application-specific transmission control method is adopted by a stream source to derive the control decisions which will be expressed through the interface to the DCCP/TP stack.

3.4.1 The Interface for Controlling the Transmission of A Stream

The interface provides a set of APIs to allow a stream source to express its transmission control decisions to DCCP/TP. Although both DCCP/TP and the real-time stream sources run at the user space, they are encapsulated in different processes or threads. Hence, an interface is needed to allow a stream source to access the buffer that holds unacknowledged packets. The interface is maintained by an independent process/thread and can be accessed by both DCCP-TP and the stream source. This interface facilitates implementing new control algorithms without the need of configuration. Configurations are typically needed in previous designs of user-level congestion control mechanisms, *e.g.*, in [14].

3.4.2 The Application-Layer Transmission Control Method

The adoption of an application-layer transmission control method depends on the nature of an application. In general, the goal of an application-layer transmission control method is to prioritize the transmission of packets that are favored by the application. Prioritizing the transmission of packets can be conducted roughly in three ways: 1) reordering the sequence of unacknowledged packets to transmit favorable packets first; 2) cancelling the transmission of those unacknowledged packets which have become insignificant to the correct functioning of an application; and 3) renewing the content of the unacknowledged data packets with updated data. In this paper, we focus on the control method which follows the second approach set forth above. We describe a deadline-based selective retransmission control method. A stream source associates each packet with a deadline when it submits the packet to DCCP/TP for transmission. An expired packet is meant to be given up its continued transmission if it remains unacknowledged by the stream sink. A stream source favors the continued transmission of important packets by assigning relaxed deadlines, while making the insignificant packets to expire soon. Moreover, whenever there are too many unacknowledged packets queued within the DCCP/TP stack, the controller is called to cancel those insignificant packets.

3.5 The Threading Structure

The transport functionality supports simultaneously delivering multiple real-time data streams. The components of the transport functionality have to be appropriately encapsulated into threads or processes. Both a stream source and a stream sink are multithreaded programs. Most of the transport functionality is on the stream source side. We have tried very hard to make the threading structure on the stream source side to be correctly arranged, such that streams will not lock up with each other.

3.5.1 On the Stream Source Side

The transport functionality on the stream source side is categorized into *system-wide service routines* and *per-stream service routines*. The *system-wide service routines* include

- **DCCP/TP Transport Service**: The general functions of DCCP/TP have been described in Section 3.2. DCCP/TP maintains a separate buffer space for each stream. All the unacknowledged outgoing packets of a stream are held in the buffer space designated for this stream. Meanwhile, DCCP/TP acts as the concierge which accepts incoming packets for all the streams and hands over the incoming packets to the stream to which the packets belong.
- **Real-Time Clock Timing Service**: It handles the timing requests for all streams.

The *per-stream service routines* include

- **Stream Emulator**: It generates data packets at their prescribed timestamps and feeds them to DCCP/TP for transmission. Whenever a stream's buffer space holding the unacknowledged outgoing packets is filled up, it means that DCCP/TP can not shoot out packets promptly. The most likely cause to this situation is that the network becomes congested. Hence, the transmission controller of this stream is called to hopefully cancel the transmission of some insignificant packets.

- **Transmission Controller**: It marks *Cancellation* on packets which should be given up their continued transmissions.

3.5.2 On the Stream Sink Side

The transport functionality on the stream sink side can also be categorized into *system-wide service routines* and *per-stream service routines*. The *system-wide service routines* include **DCCP/TP Transport Service** and **Listener**. The Listener accepts all incoming requests for establishing DCCP connections. Once a connection is established, a new thread/process is spawned which handles the data transmission of this connection. Hence, a per-stream sink handler is used to perform the application-layer handling of a stream.

4 The Deadline-based Selective Retransmission Method

We describe a transmission control method which selectively retransmits data packets based upon their deadlines. Each packet is assigned with a deadline when it is submitted to DCCP/TP for transmission. Once the temporary buffer used by a stream for holding up unacknowledged data packets is filled up, the transmission controller of this stream is invoked to cancel the continued transmission of the expired packets.

4.1 Deadline

The deadline of a data packet is expressed as a time interval which starts at the time when the packet is submitted to DCCP/TP stack and ends at the time when the packet becomes invalid. Deadlines can be assigned in various ways depending on the nature of a real-time streaming application. In this papers, we only want to demonstrate that the deadline-based selective retransmission method helps the timely delivery of packets by maintaining a low transmission delay. Hence, we consider assigning deadlines that are derivatives of the round-trip times (RTTs). We consider adopting the smoothed RTTs (SRTTs) which are evaluated based on the raw RTTs. When a sequence of RTTs is denoted by $\{r_1, r_2, \cdots, r_k, \cdots\}$, the sequence of SRTTs is denoted by $\{s_1, s_2, \cdots, s_k, \cdots\}$ with $s_k \leftarrow s_{k-1} \cdot (1-\alpha) + r_k \cdot \alpha$, $(0 < \alpha < 1)$. α is a parameter which determines the degree of the smoothness. Under a large value of α, s_k's reflect more of the dynamics of r_k's. Under a smaller value of α, s_k's are less affected by dynamic r_k's. The DCCP/TP stack updates its SRTT value whenever an RTT sample is derived. In this paper, we demonstrate the ability of the deadline-based selective retransmission method by assigning deadlines as $2 \cdot \mathrm{SRTT}$.

4.2 Management of Unacknowledged Packets

DCCP/TP on the source side maintains a list of unacknowledged packets for each stream. Every unacknowledged packet has its record in the list. A record includes the sequence number, the time when a packet was submitted to the DCCP/TP stack, and the deadline of the packet. The list will be searched when an acknowledgment packet is received or a transmission control action is performed. The list structure has to be carefully defined in order to facilitate list searches. In our work, the list is organized as a double list. The double list consists of two single lists. One single list is called the *ACK List* which maintains the acknowledgment related information, *e.g.*, sequence number and acknowledgment status. The other single list is called the *REMDL List* which maintains deadline related information, *e.g.*, the remaining time before the expiration of a packet. An example double list is illustrated in Figure 1. Each unacknowledged data packet has a record in each of the two single lists. The *ACK List* is scanned when an acknowledgment packet is received. The *REMDL List* is scanned when the transmission controller tries to cancel the expired packets. Once a packet is acknowledged or removed, its records in both single lists are deleted. Elements in either of the two single lists are sorted in order. The reason that a double list is adopted is justified as follows. A packet could be in different positions in a list when the list is sorted under different criteria. Thus, separate sorted single lists are maintained.

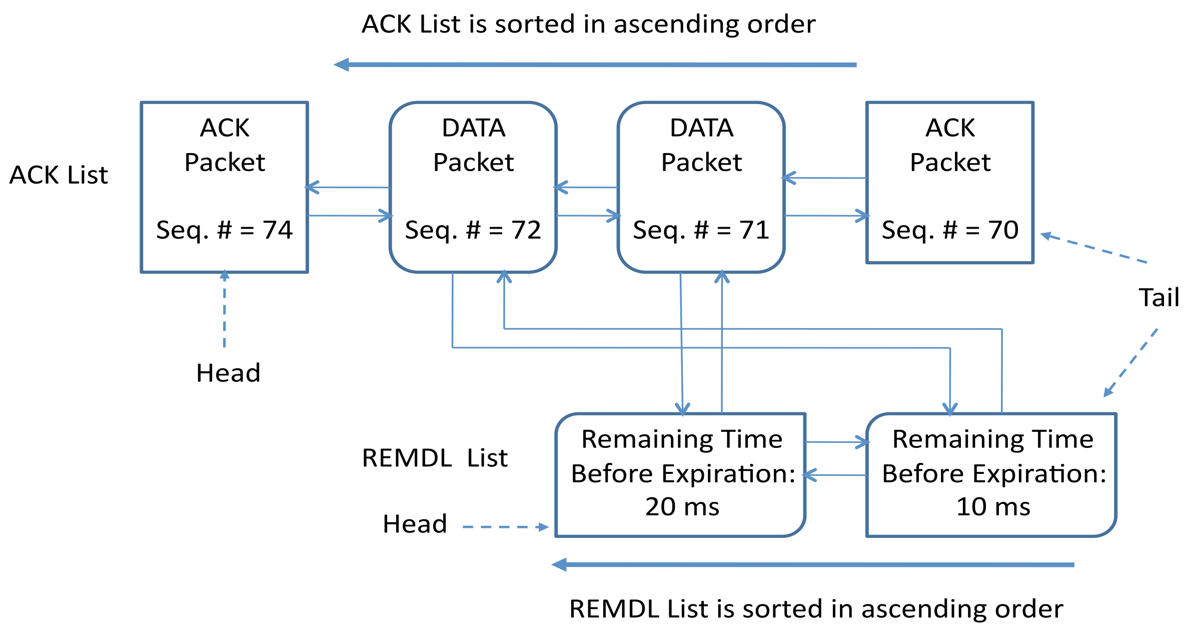

ACK List is sorted in ascending order
ACK List
ACK Packet Seq. # = 74
DATA Packet Seq. # = 72
DATA Packet Seq. # = 71
ACK Packet Seq. # = 70
Head
Tail
REMDL List
Head
Remaining Time Before Expiration: 20 ms
Remaining Time Before Expiration: 10 ms
REMDL List is sorted in ascending order

Figure 1: An illustration of a sample double list.

4.3 Selective Retransmission

When the transmission controller is invoked, it scans the *REMDL List* to detect expired data packets which remain unacknowledged. Since the *REMDL List* is a sorted list, the controller starts its scan from the packet with the earliest deadline and moves toward the packet with the latest deadline. Once the transmission controller decides to cancel the continued transmission of a data packet, it changes the acknowledgment state of this packet from *Unacknowledged* to *Cancelled*. The controller stops scanning the *REMDL List* whenever it meets the first data packet which is not expired. In the DCCP protocol, all the acknowledgment packets are reliably delivered, so the controller does not mark cancellations for acknowledgement packets. Meanwhile, this paper only describes an example method of cancelling the continued transmission of unacknowledged data packets. In this example control method, the controller marks all expired unacknowledged data packet for cancellation. The controller can also adopt other method.

The controller only labels cancellation marks on the records of packets. The records of cancelled packets will be physically removed from the double list by the DCCP/TP stack. When the DCCP/TP stack receives a new incoming packet for a stream, it scans the list of unacknowledged packets of this stream. DCCP/TP removes the records of cancelled packets from the list. An application can selectively retransmit unacknowledged data packets by adopting an appropriate transmission control method. As long as an unacknowledged data packet is not cancelled by the controller, DCCP/TP keeps retransmitting it whenever it is possible.

5 Experimental Evaluation

We demonstrate the ability of maintaining a low and stable transmission delay by adopting a deadline-based selective retransmission control method. A set of experiments are run in a network whose topology is illustrated in Figure 2. 7 stream emulators are run at the stream source, and all the 7 streams are destinated to the same stream sink. All the streams are real-time streams with a 10 ms time spacing between two consecutive packets of the same stream. Each stream source transmits 5000 packets. Packet losses are introduced at the center router shown in Figure 2. Due to the packet losses, some packets have to be retransmitted if there are designated to be reliably transmitted. We demonstrate the ability of our deadline-based selective retransmission control method on maintaining a low and stable transmission latency, as well as maintaining the dispersion between packets of the same stream.

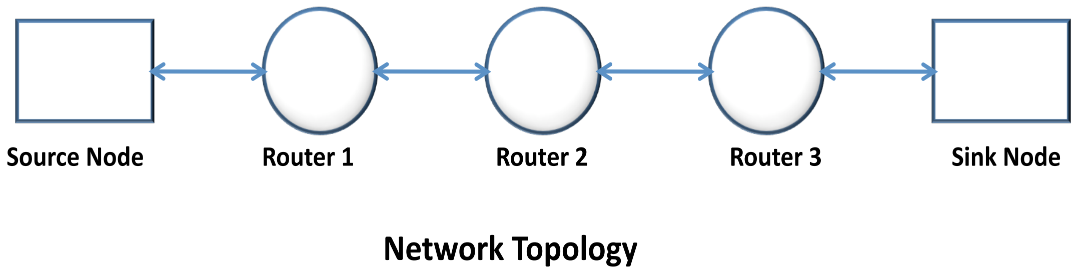


Figure 2: The Topology of the Sample Network Used in the Experiments.

The ability of maintaining a low and stable latency is demonstrated by comparing the latency when controlled under the deadline-based selective retransmission control method with the latency when controlled under a TCP-like reliable transmission control method. We compare both the round-trip times (RTTs) and the transmission latency that are measured on the source side. The transmission latency is the time interval from the moment when a packet is submitted to the DCCP/TP stack for transmission till the packet is acknowledged. The transmission latency spent in delivering a packet is simply a sum of the RTT experienced by the packet and the waiting time that the packet spent inside the DCCP/TP stack. The transmission latencies are compared by their 90% confidence interval as shown in Figure 3 (1). A 90% confidence interval demonstrate both the average of a set of sampled values of a metric and a range which includes 90% of the sampled values. A narrow range exhibits that a metric is stable. Under the deadline-based selective retransmission control method, the average value of transmission latencies is smaller, and the sampled values of transmission latencies are distributed in a relatively narrower range (ref. Figure 3 (1a)). Under the TCP-like reliable transmission control, the transmission latencies are relatively longer and less stable (ref. Figure 3 (1b)). A similar observation can also be made on the RTT samples. The deadline-based selective retransmission control method maintains much more stable RTTs than the TCP-like reliable transmission control method (ref. Figure 3 (2)).

Maintaining a stable dispersion between consecutive packets on the stream sink side is also important to many real-time streaming media applications. Stable packet dispersions facilitate appropriately scheduling data processing on the stream sink side. The deadline-based selective retransmission control method maintains much stable packet dispersions than the TCP-like reliable transmission control (ref. Figure 3 (3)). Moreover, the average values of packet dispersions on the sink side are around 20 ms, as compared to the original 10 ms packet dispersion adopted by the stream emulators. This means that packet dispersions are lengthened by network transmission.

6 Conclusions

This paper explored a deadline-based selective retransmission control method. This method targets to maintain low and stable transmission latencies by allowing a real-time streaming media application to cancel the continued transmission of expired packets. An application can favor the reliable transmission of important packets by allowing them to be transmitted for extended periods. Meanwhile, the insignificant packets can be removed from continued transmission after their expiration. Through network experiments, we have demonstrated that transmission latencies of packet can be maintained low and stable when the transmissions of real-time streams are under the control of a deadline-based selective retransmission control method.

References

[1] Eddie Kohler, Mark Handley, and Sally Floyd. Datagram Congestion Control Protocol (DCCP). In *Miscellaneous*, 2006. IETF RFC 4340.

[2] Tom Phelan. DCCP-TP—a fresh-start implementation of the Datagram Congestion Control Protocol (DCCP).

[3] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Von Jacobson. RTP: A Transport Protocol for Real-Time Applications. In *Miscellaneous*, 2003. Internet proposed standard RFC 3550.

[4] Patrick G Bridges, Gary T Wong, Matti Hiltunen, Richard D Schlichting, and Matthew J Barrick. A configurable and extensible transport protocol. In *ToN*, 15(6):1254–1265, Dec 2007.

[5] Haryadi S Gunawi, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Deploying safe user-level network services with icTCP. In *Proceedings*, pages 317–332, 2004.

[6] David Ely, Stefan Savage, and David Wetherall. Alpine: A User-Level Infrastructure for Network Protocol Development. In *Proceedings*, pages 171–184, 2001.

[7] Tatsuo Nakajima and Hideyuki Tokuda. User-level Real-Time Network System on Microkernel-based Operating Systems. In *RTS*, 14(1):45–60, Jan 1998.

[8] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-Net: a user-level network interface for parallel and distributed computing. In *Proceedings*, pages 40–53, 1995.

[9] Torsten Braun, Christophe Diot, Anna Hoglander, and Vincent Roca. An Experimental User Level Implementation of TCP. In *Techreport*, (RR-2650), Sophia Antipolis, France, Sep 1995.

[10] Aled Edwards, Greg Watson, John Lumley, David Banks, Costas Calamvokis, and C Dalton. User-space protocols deliver high performance to applications on a low-cost Gb/s LAN. In *Proceedings*, pages 14–23, 1994.

[11] Suchitra Raman, Hari Balakrishnan, and Murari Srinivasan. An Image Transport Protocol for the Internet. In *Proceedings*, pages 209–219, 2000.

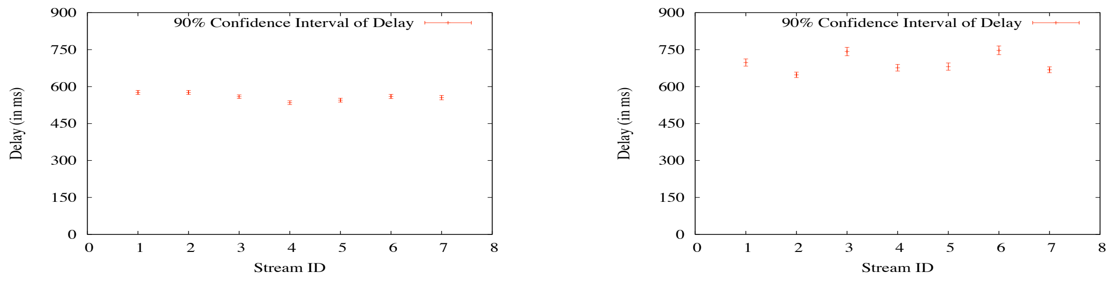
[12] NLANR. Iperf Bandwidth Measurement Toolkit. May 2005.

[13] Ingo Molnar. The Real-Time Linux Wiki and Kernel Patch. In *Miscellaneous*.

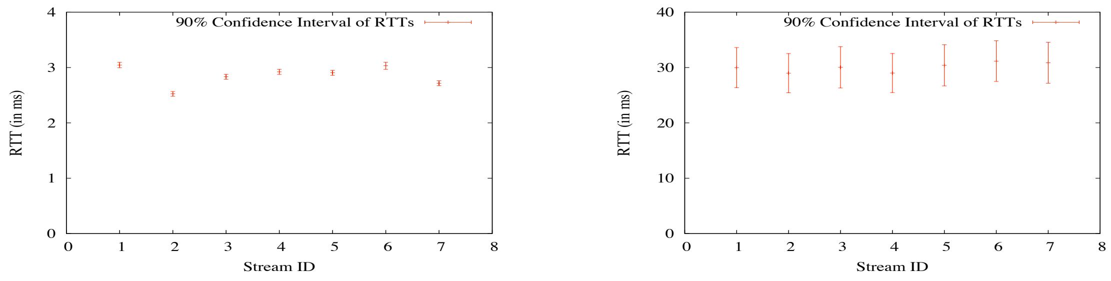
[14] Yunhong Gu and Robert L Grossman. Supporting Configurable Congestion Control in Data Transport Services. In *Proceedings*, pages 31–41, 2005.

(a) With Deadline-based Selective Retransmission Control

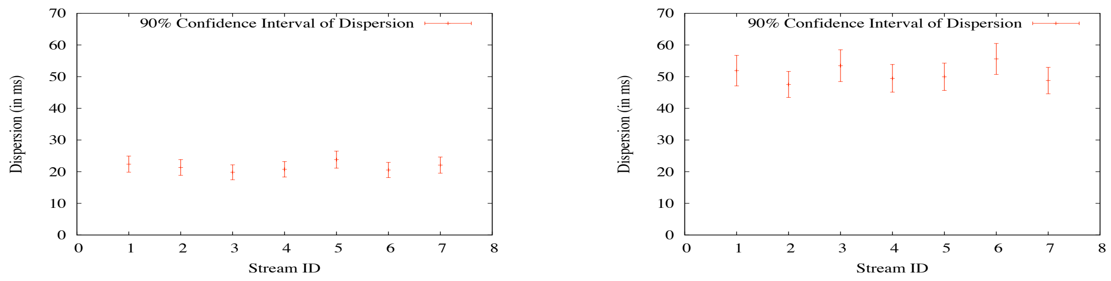
(b) TCP-like Reliable Transmission



(1) 90% Confidence Interval of Transmission Latencies (Measured on the Source Side)



(2) 90% Confidence Interval of Round-Trip Times (Measured on the Source Side)



(3) 90% Confidence Interval of Dispersions (Measured on the Sink Side)

Figure 3: Illustration of the benefit of the deadline-based selective retransmission control method on maintaining stable transmission delays for packets in real-time streams. The illustration is made by comparing the transmission delays experienced by packets with and without the selective retransmission control. The packet delay experienced by a packet is measured as the time interval from the packet is submitted to the DCCP/TP stack till the packet is acknowledged. The dispersions between packets are measure as the time interval between two consecutive packets in the same stream on the stream sink end.