International Journal of Electronics and Communication Engineering (IJECE)
ISSN 2278-9901
Vol. 2, Issue 2, May 2013, 105-114

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

MODIFIED SACK-TCP AND SOME APPLICATION LEVEL TECHNIQUES TO SUPPORT REAL-TIME APPLICATION

SYED SAMSUL AREFIN, IBRAHIM AZAD & HUMAYUN KABIR

Department of Computer Science and Telecommunication Engineering, Noakhali Science and Technology University, Sonapur, Noakhali, Bangladesh

ABSTRACT

From the very beginning of the Internet, transport layer protocol TCP is generally not used for multimedia applications because it is optimized to provide reliable delivery of all data, rather than timely delivery of sufficient data. But to communicate in real time fashion a lot of application-specific and candidate real-time transport protocols such as UDP, SCTP are arrived but no one of them has achieved that maturity and acceptance of TCP. In this paper, we investigate exactly what changes should TCP employ to support real time communication. We also find some application level technique to support Modified TCP to operate well over interactive real-time application. The validation of the Modified TCP under the most widely used flexible network simulator NS2 shows that it allows the real-time application to operate very well, up to 4% of network drop rate. That is the Modified TCP along with some application level technique is superior then UDP and regular TCP.

KEYWORDS: Real-Time Communication, SSTHRESH, SACK-TCP

INTRODUCTION

TCP [1], a process-to-process connection-oriented, reliable, having flow and error control mechanisms transport protocol that creates a virtual connection between two processes to send data. Real-time communications (RTC) is any mode of telecommunications in which all users can exchange information instantly or with negligible latency. In this context, the term "real-time" is synonymous with “live”. Years of extensive network researches has resulted in various improvements to the TCP and its implementation, including slow-start, congestion back-off and fast retransmit [2]. As a result of this work and various tuning of the implementation, TCP is fast, efficient and responsive to network congestion conditions. A lot of measurement indicates TCP is now used for 85 to 95 percent of the wide-area Internet traffic [3]. With the current development of TCP hardware accelerators in network interface cards, TCP can be expected to be favored for more and more applications [4]. By conventional wisdom, TCP is not suitable for such type of traffic because it favors reliable delivery over timely delivery which is the wrong trade-off for real-time traffic. Consequently, almost all real-time applications have been built using UDP. In this case TCP is only used as a compromise to enable communication through firewalls. The real-time applications often use a significant size of playback buffer in the order of hundreds of milliseconds or even larger in the case of simplex streaming, masking the network jitter to the application. These streaming real-time applications are in fact more sensitive to loss than jitter, at least within bounds, TCP error recovery provides significant benefit in case of packet loss. With all the clear benefits for using TCP it seems attractive to investigate whether there are modest TCP-level modifications and application-level techniques that would allow real-time applications to use TCP instead. In this thesis, we report on our exploration of techniques to allow TCP to be used for real-time streaming applications such as audio and video delivery, including interactive applications. We conclude that these techniques, including providing a real-time mode extension to TCP, make its use superior to the conventional UDP based approach for

these real-time applications. In addition, TCP is much easier to deploy than introducing new transport protocol to the internet.

RELATED WORK

There have been several recent proposals which challenge the long-held belief that TCP is unsuitable for streaming applications.

T. Nguyen and S.-C. Cheung et al. [5] propose the multiple TCP system, a receiver-driven, TCP-based system for multimedia streaming over the Internet. Their proposed algorithm aims at providing resilience against SHORT TERM insufficient bandwidth by using MULTIPLE TCP connections for the same application.

D. McCreary, Kang Li, Scott A. Watterson, David K. Lowenthal et al.[6] proposes a novel receiver-centered TCP (TCP-RC) which is a TCP modification at the receiver that is intended for delay-sensitive applications. Their basic principle is to achieve low latency at the expense of reliability.

Liang et al. [7] propose some modifications to New Reno TCP to make it more suitable for real time applications. Their approach, known as TCP-RTM, allows the receiving application to read out-of-order packets rather than waiting for lost data to be retransmitted.

B. Mukherjee et al. [8] introduces Time-lined TCP (TLTCP). TLTCP is a protocol designed to provide TCP-friendly delivery of time-sensitive data to applications that are loss-tolerant, such as streaming media players. It tries to mainly extend the sender side of TCP to deliver streaming media. It associates each segment with a deadline, and the segment will be discarded if it is still not delivered when the deadline expires.

To solve the transport problem of real-time data, a number of new protocols, such as SCP[9], SCTP[10], HPF[11] etc. have been proposed to address various aspects of transport delivery. Though they have experimented with some interesting techniques and helped us better understand the real-time data delivery problem, they have not achieved the efficiency, maturity and robustness of TCP, and their deployment will be non-trivial. It is focused on extending existing protocols, in this case TCP, to meet expanded application requirements. New Reno TCP [12] and Time-lined TCP [8] concur with on this general approach. Time-lined TCP tries to mainly extend the sender side of TCP to deliver streaming media. It associates each segment with a deadline, and the segment will be discarded if it is still not delivered when the deadline expires. This approach can work together with modified TCP and deserves further investigation. However, in general this approach seems more complicated because it requires a new TCP option attached to every TCP segment to inform the receiver the next sequence number to expect. The conventional approach of using UDP for real-time communication has led to network problems in which UDP-based applications not sensitive to network congestion problems have caused problems for other applications, particularly the TCP-based applications. Here, TCP backs off in response to the excessive UDP traffic, sometimes signaled through an active queue management mechanism such as RED, allowing UDP traffic to deny service to TCP-based applications. So to ensure secure and reliable real time communication TCP needs to be reorganized. Here we modify SACK TCP instead of Tahoe or New-Reno since it uses maximum communication bandwidth than others [13].

SACK TCP

Sack Modification to TCP [14] shows that Sack retains the basic principle of Tahoe, such as slow starts and the coarse grain re-transmit timer like New Reno[12]. However it adds some intelligence over it so that lost packets are detected earlier and the pipe line is not emptied every time a packet is lost. Reno requires that we receive immediate

acknowledgement whenever a segment is received. The logic behind this is that whenever we receive a duplicate acknowledgment, then this duplicate acknowledgment could have been received if the next segment in sequence expected, has been delayed in the network and the segments reached there out of order or else that the packet is lost. If we receive a number of duplicate acknowledgements than that means that sufficient time have passed and even if the segment had taken a longer path, it should have gotten to the receiver by now. There is a very high probability that it was lost. So Reno suggests an algorithm called 'Fast Re-Transmit' [2]. Figure-1 shows an overview of TCP fast retransmission. If there are 'w' segments in the window and one is lost, then we will receive (w-1) duplicate ACK's. Since CWND is reduced to W/2, therefore half a window of data is acknowledged before we can send a new segment. Once we retransmit a segment, we would have to wait for at least one RTT before we would receive a fresh acknowledgement. Whenever we receive a fresh ACK we reduce the CWND to ssthresh. If we had previously received (w-1) duplicate ACK's then at this point we should have exactly w/2 segments in the pipe which is equal to what we set the CWND to beat the end of fast recovery. Thus we don't empty the pipe, we just reduce the flow. We continue with congestion avoidance phase of Tahoe after that. TCP with Selective Acknowledgments (SACK) is an extension of TCP Reno and it works around the problems face by TCP Reno and TCP New-Reno, namely detection of multiple lost packets, and retransmission of more than one lost packet per RTT. SACK retains the slow-start and fast retransmits parts of RENO. It also has the coarse grained timeout of Tahoe to fall back on, in case a packet loss is not detected by the modified algorithm. SACK TCP enquires that segments not be acknowledged cumulatively but should be acknowledged selectively. Thus each ACK has a block which describes which segments are being acknowledged. Thus the sender has a picture of which segments have been acknowledged and which are still outstanding. Whenever the sender enters fast recovery, it initializes a variable pipe which is an estimate of how much data is outstanding in the network, and it also set CWND to half the current size. Whenever the window goes smaller than the CWD window it checks which segments are not received and send them. If there are no such segments outstanding then it sends a new packet [15]. Thus more than one lost segment can be sent in one RTT.

TCP MODIFICATION

To make TCP suitable for real time communication sender side application write calls and receiver side application read calls should never be blocked. The most significant modification of TCP is about packet reception.

- Out of order packet to wait queue: On reception of an out-of-order packet with a sequence number logically greater than the current receive pointer and with a reader waiting on the connection, the packet data is delivered to the waiting receiver, the receive pointer is advanced past this data and this new receive pointer is returned in the next acknowledgment segment.

- Skip the lost segment: If there is no in order segment queued in TCP connection to read by the application but one or more out-of-order segments are queued, than the first contiguous range of out-of-order segments are moved from the out-of-order queue to the application receive queue, the receive pointer is advanced over these packets, and the resulting data delivered to the application layer.

These two major modifications with some application level technique will make the modified TCP fit for the real-time application.

TCP Framing

A new framing mode is enabled that provides framing support by handling the data in each application-layer write as a separate TCP segment, maintaining these packet boundaries even on retransmission, and returning a single TCP

segment in response to each read operation on the receiver end. By writing to the TCP connection for an integral number of application layer frames, the sender can allow the receiver read not ever to receive a partial application-level frame, even with packet loss. That results application level frame boundaries always align with TCP segment boundaries. For applications with fixed-sized frames, typically true for audio applications, the following method is generally sufficient. The TCP NODELAY socket option is used at the sender to turn off the Nagle algorithm, so that each application level frame is transmitted immediately after it is written by the sender application (this avoids delay and avoids combining the frame with another frame in a TCP segment). In addition, the TCP MSS (Maximum Segment Size) is set in such a way that each TCP segment contains one or integral multiple number of application level frames. Consequently, each TCP packet loss corresponds to one or integral number of application-level frame loss. At the receiver, each read request uses the fixed frame size as the size of the read. Therefore, each read request gets one application-level frame. Further application-level sequencing can make the packet loss evident to the receiver. For instance, the RTP sequence number can be used for this purpose with TCP applications using RTP [15]. Note that this is not an extra burden on the TCP applications because UDP-based applications are already doing similar things to deal with missing frames.

TCP Sender

In general, the sender's send buffer should be large enough to accommodate all the data segments that have not been acknowledged and that are to be transmitted.

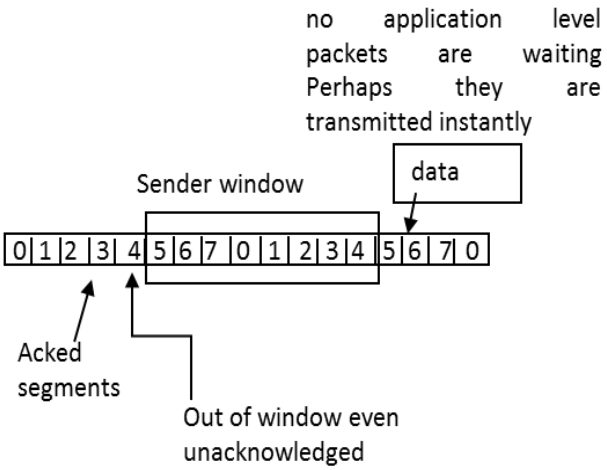

Figure 1: Sender Side Modified TCP Windowing Technique

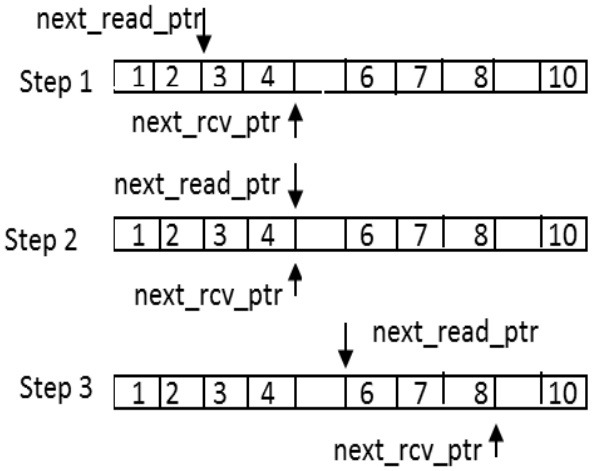

Figure 2: Modified TCP Skip-Over-Hole Technique

However, in the case that the sender's send-buffer is full due to large amount of backlogged data, TCP discards the oldest data segment in the buffer and accepts the new data written by the application and also advances its send-window past the discarded data segment. This way, the application write calls are never blocked and the timing of the sender application is not broken. In case of Regular TCP's send buffer is full it never advances its send window until the segments are acknowledged though more application data are waiting to be transmitted. In this case modified TCP sender works like Figure-1. It ensures the transmission of most updated data to receiver by advancing its send window and discarding the oldest segment even if it is not acknowledged.

TCP Receiver

At the receiver side, since TCP uses the skip-over-hole technique, old packets that are discarded by the sender will not cause any problem, and will be skipped-over by the receiver. The number of receiver-skipped-over packets will be reported to the sender by the SNACK option. The value of the loss-counter can be queried by the sender application and

used by the application to adjust its encoding scheme and transmission rate to the network condition. Figure-2 shows that segment 3 to 6 is the first contiguous segments that arrived successfully. The TCP receiver pointer *next_rcv_ptr* is waiting for segment 5even 6 and further segments are arrived while application is reading segment 2. Regular TCP will never skip segment 5 hole. In modified TCP behavior next_rcv_ptr will wait for 5 till next_read_ptr is in the same place. While both of the pointers are at the end of 4 and 5 still not arrived next_rcv_ptr has no choice but to move on to 10 segment and next_read_ptr to 6 since real-time application need most recent data to be delivered. Sender TCP needs to be notified about these phenomena, to stop unnecessary retransmission of this segment. So receiver TCP sends SNACK (selective negative acknowledgement) back to the source.

TCP Playback Buffer

Real-time streaming applications use a playback buffer to insulate the playback from jitter arising from the variable delay through the Internet. A receiver application normally delays processing to operate with a playback delay of *pbd* milliseconds later than the source so that a packet is only late in its processing if it is delayed by more than *pbd* milliseconds. According to [4], utilizing the help of fast retransmission, receiver thread may be programmed with

$$pbd \geq (3/2)*RTT + 3*period + delta, \quad \text{--- ---} \qquad (1)$$

Where RTT is the TCP round-trip time and period is the typical interval between transmissions of data on the stream and delta is some modest extra time. For instance, for a typical audio stream with period =20ms, RTT=60ms and delta=10ms, then $pbd \geq 160$ms. With the playback delay set as above, TCP can recover from a packet loss by fast retransmit within the playback delay. In particular, after a data packet loss, the receiver receives the next 3 packets as out of order packets, causing 3 duplicate ACKs to be sent back to the sender. The sender thus receives the third duplicate ACK after RTT+3*period, causing it to retransmit the last unacknowledged portion of the stream, which corresponds to the dropped packet. Moreover, the application is oblivious to the packet loss, providing better quality play out. In audio and video applications, especially with content compression, packet loss can seriously degrade the play-out quality otherwise. Even a 1% loss rate, especially on important packets, such as MPEG I frames, can make the result intolerable for video streaming applications. Burst drop from tail drop in overflowing network queues is well known as a behavior to avoid.

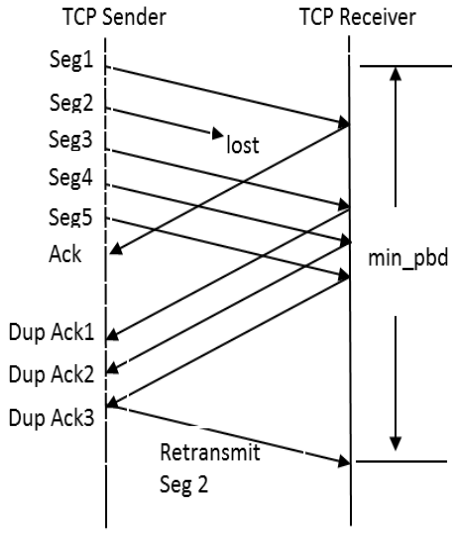

Figure 3: Evaluation of Minimum Playback Delay

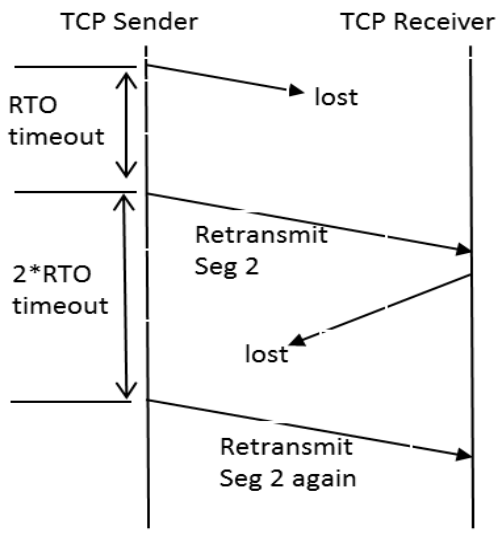

Figure 4: Unnecessary 2nd Retransmission and Long Delay

Single packet loss is the most common drop situation in a well-managed network because it is commonly usedto signal congestion to TCP flows that are ramping up to maximum throughput.

APPLICATION LEVEL HEARTBEAT

When the ACK to the retransmitted packet is lost, the sender cannot proceed and has to time out and retransmit that packet again, causing unnecessary 2nd retransmit and long delay. A TCP real time application is programmed to periodically send an application-level packet back to the source if it has not received any data within the heart beat time period. The heart beat period *hbp* is set to *hbp=3* period* ensuring that, if a single ACK is lost, an extra ACK is generated by the heartbeat packet. It prompts TCP to piggyback the ACK information that reassures the source that the data are being received and keeps the data stream flowing. It can also make sure the sender can increase its congestion window size in a timely fashion during slow start. In addition, it can also carry application-level information, such as an indication from the receiver on the amount of loss it is observing, providing the source with additional information on which to act. For instance, the source may change its encoding and report the loss behavior to the user. The application can easily check for receipt of new (in-order) data by querying the TCP buffers of its connection each time that it wakes up. Because the heart beat is only sent when the receiver has not received new in-order data, it imposes low overhead on the network in the expected case of low packet loss and, as a small packet, it imposes minimal percentage overhead in any case. In the uncommon case that both the forward path and the reverse path are losing packets, the loss of some critical ACKs may cause unnecessary delay on the data sender side.

SIMULATION

We simulated this modification in ns2 simulator. We made a network of eight nodes which is shown in figure-5. We assumed that link bandwidth are much greater then application's data rate. We assigned the exact specification of real-time application and in some cases higher than the real-time application.

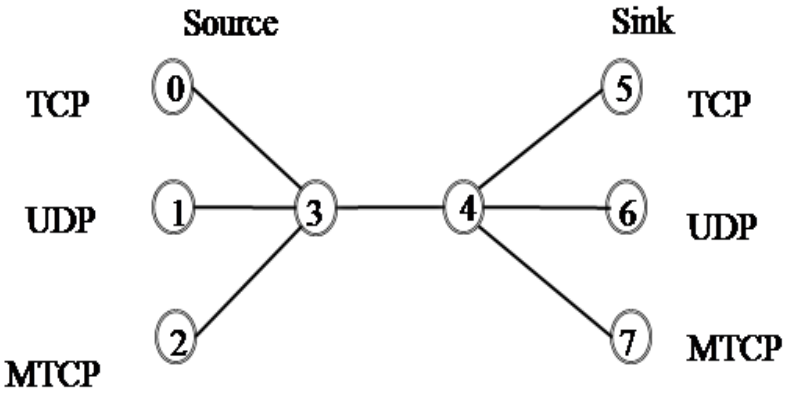

Figure 5: Simulation Topology

We used a playback delay of 250 milliseconds, which is generally considered acceptable for interactive audio applications such as teleconferencing [16]. This allows a round trip time of up to roughly 100 milliseconds, sufficiently for operation across north America in general[8]. In practice a TCP real-time application could adapt the play back delay of the actually TCP determined round trip time. Our simulations focused on single packet losses because this case is the most common. Burst losses of two or more packets are an order of magnitude less common [17]. Real-time audio and video application like teleconferencing and videoconferencing produce constant bit rate(CBR) data over the network. So we used CBR application module of ns2 where data frame stream of 100 Bytes perform at 20 milliseconds interval which reassembles the typical audio traffic. 1500 frames were transmitted in each simulation which corresponds to 100 seconds of streaming media. For interactive video application the play back delay can be higher up to 5000 milliseconds [18] which will results in even better error recovery performance.

PERFORMANCE EVALUATION OF MTCP OVER UDP

Since we know UDP has no error recovery technique, so if any packet drops it will never be recovered. Hence this will affect the application. Video is compressed as an MPEG-4 stream such that a single packet loss corrupting an I-frame can disrupt the video reception for significant fractions of a second and causes degradation in the quality of the video. But in our MTCP all lost packets have a chance to be retransmitted until the application needs that. For example let packet 1, 2, 3, 4, 5, 6, 7, 8 and 10 are to be transmitted. Let packet 1,2,3,4 and 5 has been transmitted successfully. Packet 6 has been dropped and packet 7, 8 and 9 also transmitted successfully. Also let receivers play back delay is 250 milliseconds. So, packet 6 will get 500 milliseconds to be retransmitted. If it is not retransmitted within this time the receiver has no choice but to discard it since the application is real-time. We run the simulation up to five percent drop rate.

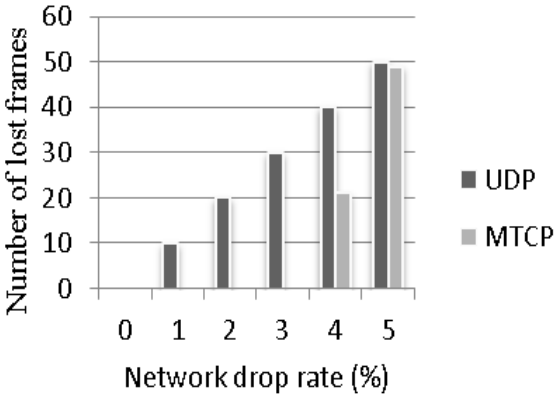

Figure 6: Network Drop Rate Vs Number of Lost Frames (RTT = 100ms)

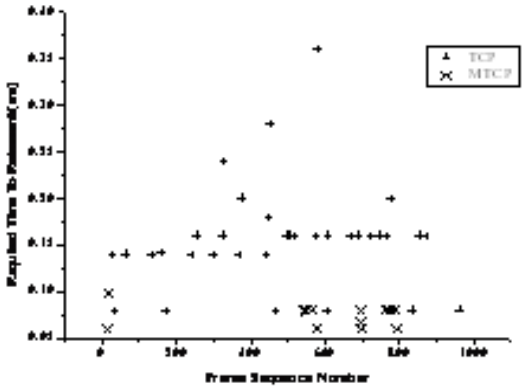

Figure 7: Frame Sequence Number Vs Retransmit Time (RTT=100ms)

Figure-6 shows that the MTCP can tolerate up to three percent of drop rate with no loss. That is the lost packets are retransmitted within the play out time while UDP does not care about this loss. When the drop rate is greater than three percent the MTCP behaves like UDP but relatively good performance then UDP. In a reasonably well managed network, it is the common case that the packet loss rate stays below four percent. As noted before, a one percent loss rate may render the quality of some real-time applications intolerable, while using just these two simple application-level techniques can enable such applications to achieve good performance, in terms of recovering all lost packets and introducing low jitter, under loss rate as high as four percent. With these two techniques, TCP real-time applications perform well in networks with modest levels of packet loss provided that the round-trip time remains relatively stable and within the range used to compute the play back delay.

PERFORMANCE EVALUATION OF MTCP OVER TCP

In this section we are going to evaluate the sack's fast-retransmit technique used in MTCP. When a packet is lost regular TCP waits for the RTO (Retransmission Time Out) to timeout for the retransmission. Typically, RTO is much greater than several play back delay. So the application will stop playing the upcoming stream until the retransmitted packet is successfully received by the receiver. Sometimes it may be several minutes. We run the simulation at three percent drop rate. Then we identified the packets that are dropped. For these lost packets we recorded the time when these are sent for the first time and for the second time and if the retransmitted packet is also lost we recorded the next sent time for that too. In summery we recorded the time when the packet is sent for the first time and last time when the packet is sent after lost. We calculated the different between them for each lost packets and plot those different against frame sequence number. Figure-7 represents the time in the Y axis. We can see that MTCP takes lower time. So in MTCP the lost

packets can be retransmitted before the application needs it. While in TCP the lost packets reached the receiver later than the application needs it. SACK retains the slow-start and fast retransmits parts of RENO. It also has the coarse grained timeout of Tahoe to fall back on, in case a packet loss is not detected by the modified algorithm. SACK TCP requires that segments not be acknowledged cumulatively but should be acknowledged selectively. Thus each ACK has a block which describes which segments are being acknowledged. Thus the sender has a picture of which segments have been acknowledged and which are still outstanding. Whenever the sender enters fast recovery, it initializes a variable pipe which is an estimate of how much data is outstanding in the network, and it also set CWND to half the current size. Every time it receives an ACK it reduces the pipe by 1 and every time it retransmits a segment it increments it by 1. Whenever the pipe goes smaller than the CWD window it checks which segments are not received and send them.

CONCLUSIONS

Modified TCP provides a new mode for TCP that allows it to support a range of real-time applications with performance superior to using a basic datagram service such as UDP. Modified TCP makes all the benefits of TCP available to these real-time applications while allowing application developers to use one transport protocol, not two or several. It also ensures real-time applications that are responsive to network congestion in a way compatible TCP provides a new mode for TCP that allows it to support a range of real-time applications with performance superior to using a basic datagram service such as UDP with established TCP dynamics. Several aspects of this work are of note. First, it is showed that the Modified TCP receiver behavior of stepping over missing packets and the sender behavior of advancing the window independent of acknowledgments ensured a real- time application was seldom stalled by excessive levels of packet drop. Second, the structuring an application to use the TCP buffering for the playback buffer together with fast retransmit allows retransmission on packet loss without introducing jitter at normal levels of network packet drop. The resulting recovery of lost packets is important with applications such as video and audio streaming that are made loss sensitive by their use of compression techniques. Finally, the minor modification to the SACK implementation enables TCP to provide congestion response that is compatible to regular TCP, whether in real-time mode or not. Overall, in conclusion, it is a better engineering decision to extend TCP to support real-time than to require real-time applications to use a separate specialized transport protocol, given the many benefits of using TCP and not having to support and use a separate protocol. Modified TCP is a promising definition of such an extension and hope to see it incorporated in the standard implementation of TCP.

REFERENCE

1. J.B. Postel, *"Transmission Control Protocol"* RFC 793, Information Sciences Institute, September 1981.
2. W.R. Stevens, "*TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms"*, Jan 1997, RFC 2001.
3. K. Claffy, Greg Miller, and Kevin Thompson, *"The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone",* In Proceedings of INET, 1998.
4. K. Thompson, G. Miller and R. Wilder, *"Wide-Area Internet Traffic Patterns and Characteristics"*, IEEE Network, 11(6), pp. 10-23, Nov. /Dec. 1997.
5. T. Nguyen and S.C. Cheung, *"Multimedia Streaming Using Multiple TCP Connections"*Journal ACM TOMCCAP Jan 2008.

6. D. McCreary, Kang Li, Scott A. Watterson, David K. Lowenthal, *"TCP-RC: A Receiver-Centered TCP Protocol for Delay-Sensitive Applications."* SPIE Dec 2005.
7. S. Liang and D. Cheriton, *"TCP-RTM: Using TCP for Real Time Multimedia Applications", ICNP*, 2002.
8. B. Mukherjee and T. Brecht, *"Time-Lined TCP for the TCP-Friendly Delivery of Streaming Media,"* in Proceedings of IEEE ICNP '00, November 2000.
9. S. Cen, C. Pu and J. Walpole, "*Flow and Congestion Control for Internet Media Streaming Applications"*proceedings of Multimedia Computing and Networking, 1998.
10. R.Stewart and Q.Xie, *"Stream Control Transmission Protocol"*, December 2000, RFC2960.
11. Jia-Ru Li, Sungwon Ha and VaduvurBharghavan, *"HPF: A Transport Protocol for Supporting Heterogeneous Packet Flows in the Internet"*, Infocom 1999.
12. S.Floyd and T.Henderson*"The New-Reno Modification to TCP's Fast Recovery Algorithm"*RFC 2582, Apr 1999.
13. K. Fall and S. Floyd. "*Simulation-based Comparisons of Tahoe, Reno, and SACK TCP*." Computer Communication Review, 26(3):5-21, July 1996. (Pubitemid 126545922)
14. Matthew Mathis, JamshidMahdavi, Sally Floyd and AllynRomanow, "*TCP Selective Acknowledgment option*", (Internet draft, work in progress), 1996.
15. Audio-Video Transport Working Group, H. Schulzrinne, S. Casner R. Frederick and V. Jacobson, *"RTP: A Transport Protocol for Real-Time Applications",* RFC1889.
16. Xu Chang-Biao, Long Ke-Ping and Yang Shi-Zhong, *"Corruption-based TCP rate adjustment in wireless networks"*, In Chinese Journal of Computers, 2002, 25(4), pp. 438-444.
17. Georg Carle and Ernst W. Biersack, *"Survey of Error Recovery Techniques for IP-based Audio-Visual Multicast Applications",* IEEE Network 11(6), pp 24-36, December 1997.
18. Ahmed Mehaoua and Raouf Boutaba, *"The Impacts of Errors and Delays on the Performance of MPEG2 Video Communications",* IEEE International Conference On Acoustics, Speech, and Signal Processing (ICASSP 99), Phoenix, Arizona, March 1999.