DESIGN, IMPLEMENTATION AND EVALUATION OF CONGESTION CONTROL MECHANISM FOR VIDEO STREAMING

Hiroki Oda[1] Hiroyuki Hisamatsu[2] Hiroshi Noborio[2]

[1]Graduate School of Computer Science and Arts,
Osaka Electro-Communication University, Osaka, Japan
hiroki.oda@olnr.org
[2]Department of Computer Science, Osaka Electro-Communication University,
Osaka, Japan
hisamatu@isc.osakac.ac.jp, nobori@noblab.osakac.ac.jp

ABSTRACT

In recent years, video streaming services over TCP, such as YouTube, have become more and more popular. TCP NewReno, the current TCP standard, performs greedy congestion control, which increases the congestion window size until packet loss occurs. Therefore, because TCP transmits data at a much higher rate than the video playback rate, the probability of packet loss in the network increases, which in turn takes bandwidth from other network traffic. In this paper, we propose a new transport-layer protocol, called TCP Stream, that solves the problem of TCP in video streaming. TCP Stream performs a hybrid congestion control that combines the loss-based congestion control, which uses packet loss as an index of congestion, and the delay-based congestion control, which uses delay as an index of congestion. Simulation and experimental results show that TCP Stream transmits data at the adjusted rate, unlike TCP NewReno, and does not steal bandwidth from of other network traffic.

KEYWORDS

Video Streaming, TCP (Transmission Control Protocol), Congestion Control, YouTube

1. INTRODUCTION

In recent years, the video streaming services such as YouTube [1], Dailymotion [2], and Veoh [3] have become more and more popular. These services communicate using HTTP. We can easily use them to watch a video sequence using a web browser, if Adobe Flash Player is installed on our computer or our web browser supports HTML5 (the next generation of HTML). HTTP uses TCP as its transport-layer protocol.

However, TCP is not ideal for video streaming because of the way it handles congestion control [4]. The congestion control algorithm of TCP NewReno, the current standard version of TCP, controls the transfer rate by adjusting the number of data packets that can be transmitted without acknowledgment, named “congestion window size.” TCP NewReno increases its congestion window size until packet loss occurs. Therefore, TCP NewReno transfers data at a much higher rate than the video playback rate. As a result, video streaming traffic uses bandwidth from other background traffic unnecessarily. Moreover, when packet loss does occur, TCP NewReno greatly reduces the size of the congestion window, which means that the transmission rate required for the continuous playback is temporarily insufficient.

In this paper, we propose a new transport-layer protocol, TCP Stream, that solves the problems of video streaming over TCP. TCP Stream uses a hybrid congestion control that combines a loss-based congestion control and a delay-based congestion control. The loss-based congestion control uses packet loss as an index of congestion, which is similar way to TCP NewReno. The delay-based congestion control uses a network delay as an index of congestion. Using simulation results, we show that TCP Stream utilizes the network bandwidth effectively when the network has vacant bandwidth, and does not use bandwidth from other network traffic. TCP Stream transfers data at the adjusted transmission rate when a network is in a congestion state, unlike TCP NewReno. Furthermore, we implement TCP Stream on a Linux kernel, and evaluate the performance of TCP Stream in an experiment network environment and show TCP Stream transfers the video sequence according to the network congestion status.

Much research has been conducted on new transport layer protocols for video streaming [5-12]. For instance, researchers have proposed transport layer protocols that do not rapidly increase or decrease the size of the congestion window, yet remain TCP-friendly [5-7]. However, these protocols do not avoid the above-mentioned issue of having too much throughput relative to the video playback rate.

Moreover, in [8, 9], the authors propose modifications to TCP that keep the data transfer rate at the rate requested by the upper-layer applications. For instance, in [9, 13], the authors propose mechanisms to stabilize TCP throughput by concealing packet loss from TCP. They do so by installing a Forward Error Correction (FEC) layer in the lower part of the transport layer at both the sender and receiver. However, these mechanisms transmit excessive traffic over the network due to the redundancy of the FEC.

The research in [10] proposes a new transfer mechanism for video streaming over TCP. They show that the proposed mechanism achieves a low frequency of buffer underflow at the receiver, and does not steal excessive bandwidth from competing traffic. Although the performance of this mechanism is excellent, the proposed mechanism must be installed as an application program at the sender and receiver. This makes it difficult to deploy widely, because it requires users to install the application. In this paper, we propose a new video streaming transport-layer protocol that only requires the sender-side TCP to change.

In [11], the authors have proposed a client-driven video transmission scheme that utilizes multiple HTTP/TCP streams. Using Linux boxes experiments, they have shown that their scheme mitigate the throughput variations. However, their scheme requires modifying both of the sender and receiver. In [12], a new stored video streaming system has been proposed. The system discards the low priority frames by implementing an input buffer at the application-layer. Simulation results have shown that their system can efficiently stream a video sequence over TCP if the client playout delay is large. However, we strongly believe that a user does not want to decrease the video sequence quality in the stored video streaming even when the network is congested.

Also presented here are application layer protocols for media streaming [14, 15]. Windows Media Player and Real Player use the Real-Time Streaming Protocol (RTSP [14]), and the Real-Time Messaging Protocol (RTMP [15]) is a proprietary protocol of Adobe Systems. Currently, both RTSP and RTMP only allow viewing and listening to streamed media. They request data transfer from their lower protocols, and do not conduct transfer control themselves. Thus, they do not resolve the problem of streaming traffic stealing bandwidth from competing traffic.

The rest of this paper is organized as follows. First, in Section 2, we explain the congestion control algorithm of TCP NewReno, and the problems it has when transferring video sequences. We then briefly describe the features and problems of the data transfer mechanism of the current streaming service in Section 3. In Section 4, we propose a new transport-layer protocol, called TCP Stream. In Section 5, we show the effectiveness of TCP Stream by simulations and experiments in the experiment network. Finally, in Section 6, we conclude this paper and discuss future work.

2. The TCP Problem when Streaming Video

In this section, we explain the TCP NewReno congestion control algorithm, which is widely utilized on the Internet. We then explain the problems encountered when TCP NewReno is used in video streaming.

TCP NewReno controls a packet transfer rate using a loss-based congestion control algorithm, which uses packet loss as an index of the congestion status. The TCP NewReno transfer rate is adjusted with the congestion window size, which refers to the number of data packets that can be transmitted without any acknowledgement. The TCP NewReno congestion control algorithm has two phases, called the slow-start phase and the congestion avoidance phase. The increase in the speed of the congestion window size changes with each phase.

The slow-start phase operates when establishing a connection and when a timeout occurs. The congestion window size increases exponentially in the slow-start phase. If the congestion window size exceeds a threshold, called ssthresh, it shifts to the congestion avoidance phase. In the congestion avoidance phase, TCP NewReno increases the congestion window size linearly until a packet loss occurs. In other words, TCP NewReno uses all the bandwidth of the network. Although this is an appropriate control from the point of view of using the network bandwidth effectively, this can cause problems on the network. When TCP NewReno is used for video transfer, it sends data at a higher rate than the video playback rate, due to its greedy congestion control. Consequently, the packet loss probability in a network increases, and the quality of the video streaming may suffer. Moreover, the available bandwidth for other network traffic decreases, since video streaming transfers data at a high rate superfluously.

When packet loss does occur, TCP NewReno judges that the network is in a congestion state, and decreases the congestion window size. The decrease in size changes with packet loss detection methods. TCP NewReno judges that the network is in a slight congestion state when it detects packet loss after receiving three duplicate ACKs (Acknowledgements), and then reduces the congestion window size by half. Moreover, when a timeout occurs, TCP NewReno judges that the network is in a serious congestion state, and the congestion window size is decreased to one. Since the increased speed of the congestion window size in the congestion avoidance phase is one packet in one round-trip time, if the congestion window size decreases, the recovery of the congestion window size will take a long time. Therefore, when using TCP NewReno for video streaming, even if there is sufficient bandwidth in a network, the transmission rate can be temporarily less than the video playback rate, and the playback of the video can stop.

As stated above, when TCP is used for video streaming, TCP sends packets at a rate far exceeding the video playback rate, and takes bandwidth from background traffic. Moreover, there is the other problem that the transmission rate can be temporarily less than the video playback rate, causing the playback of the video to stop.

3. Current Video Streaming Mechanisms over TCP

In this section, we clarify the current features and problems encountered by data transfer mechanisms when used for video streaming services, such as YouTube. Please refer to [10] for the details of the investigation method and its results.

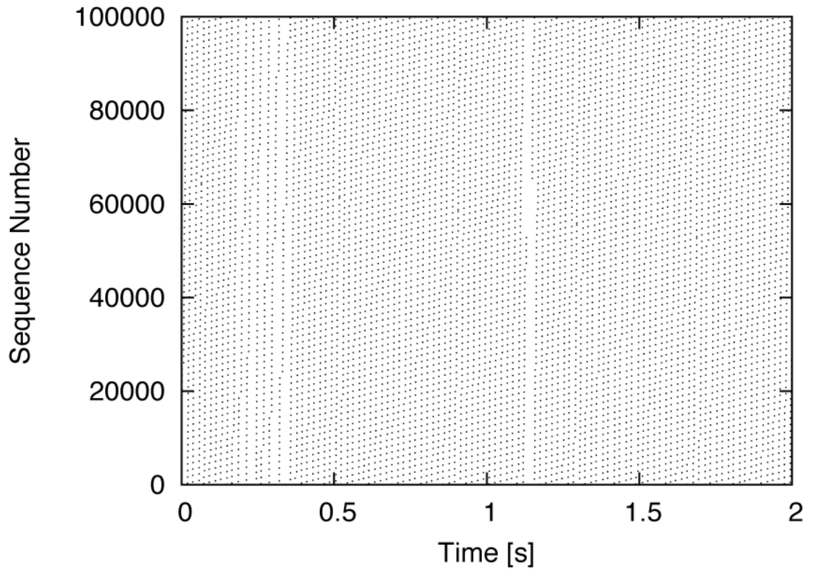


Figure 1. YouTube data packet receiving

YouTube has two mechanisms for video data transfer. We refer to these two mechanisms below as mechanism(i) and mechanism(ii), respectively. First, we explain mechanism(i), which YouTube uses in almost all cases to transfer data. Figure 1 shows the receiving timing of data packets under mechanism(i). The y axis of the graphs represents the byte-count sequence number (mod 100,000) of received TCP data packets. The x axis represents the time, which is zero when the first data packet is received. Mechanism(i) uses no special control, and the server sends video data at an extremely high rate from the beginning to the end of the video sequence. The average data transfer rate is much higher than the video playback rate.

YouTube uses mechanism(ii) to transfer a few video sequences, and is a more intelligent transfer mechanism than mechanism(i). Mechanism(ii) has two phases: a first and a second phase. In mechanism(ii), a server transmits a lot of data in the first phase, whereupon it interrupts the data transfer. Several seconds after the first phase is complete, YouTube shifts to the second phase. In the second phase, YouTube transmits data, and then switches over to discontinuous data spurts, which continue until the data transfer is complete. With mechanism(ii), YouTube transmits data at a higher rate than the video playback rate in both the first and the second phase.

In summary, YouTube transfers video data at extremely high rate relative to the video playback rate. This is good from the point of view of enabling continuous playback. However, it is a serious problem from the point of view of needless competition with other applications.

4. TCP STREAM

In this section, we propose a new transport-layer protocol for video streaming, called TCP Stream. We transfer video data by changing the congestion control algorithm of the sender-side TCP. That is, the receiver-side TCP, on the user side, does not need to change. Therefore, using this data transfer mechanism, users do not need to change the configuration of the operating system or modify the kernel, and do not need to install any special application for watching video sequences.

TCP Stream performs window-based congestion control that combines two congestion controls: a loss-based congestion control that uses packet loss as an index of congestion, and a delay-based congestion control that uses a network delay as an index of congestion. We use the following terminology: the loss window size is the value of the loss window used by the loss-based congestion control; the delay window size is the value of the delay window used by the delay-based congestion control; and the send-out window size is the value of the send-out window used for the control when sending out packets. The send-out window size is the sum of the loss window size and the delay window size.

The loss-based congestion control increases the loss window size until it uses all the network bandwidth and fills the routers' buffers, at which time a packet loss occurs. Therefore, the loss-based congestion control is called greedy congestion control. The delay-based congestion control controls the delay window size to keep the number of packets in the routers' buffers below a fixed value. Therefore, when a network is in a congestion state, namely when many packets are in the routers' buffers, the delay window size does not increase. TCP Stream conducts the congestion control by combining these two methods. TCP Stream achieves a transfer rate that it needs in order to play back the video, in other words, the video playback rate, by using the loss-based congestion control. Furthermore, when the network is not in congestion state, TCP Stream transfers data at a high transfer rate when using the delay-based congestion control.

We consider the case where TCP Stream competes with TCP NewReno. When TCP that uses a delay-based congestion control competes with TCP that uses the loss-based congestion control, the former transmission rate degrades [16]. This also applies to the delay-based congestion control of TCP Stream. When TCP Stream competes with TCP NewReno, which uses the loss-based congestion control, the delay window size of TCP Stream continues decreasing, until it becomes almost zero. Consequently, the send-out window size is determined by the loss window size used by the loss-based congestion control. The loss-based congestion control is the same control used by TCP NewReno. Therefore TCP Stream performs TCP-friendly data communication when competing with TCP NewReno.

Compound TCP [17] is an existing transport-layer protocol that combines the loss-based congestion control and the delay-based congestion control. However, Compound TCP is a transport-layer protocol for a high speed and long distance network. When Compound TCP is used for video transfer, it transfers data at a much higher rate than the video playback rate, and so, similarly to TCP NewReno, takes bandwidth from other traffic unnecessarily.

The loss-based congestion control is a greedy control that increases the window size until a packet loss occurs. In TCP Stream, a maximum value is assigned to the loss window size that a loss-based congestion control uses, so that TCP Stream may not take the bandwidth of other traffic unnecessarily. Specifically, we set the maximum value of the loss window size to the number of packets required during one round-trip in order to play back the video sequence. The loss window size, *lwnd*, is given by:

$$lwnd \leftarrow \begin{cases} min(lwnd + 1, \lceil Rate \cdot RTT/MSS \rceil) \\ \qquad \text{(slow-start phase)} \\ min(lwnd + \frac{1}{lwnd}, \lceil Rate \cdot RTT/MSS \rceil) \\ \qquad \text{(congestion avoidance phase)} \end{cases} \tag{1}$$

where *Rate*, *RTT*, and *MSS* are the video playback rate, the round-trip time of the TCP Stream connection, and the maximum segment size of TCP Stream, respectively. The video playback rate is configured on the server-side in advance. Since video sequences are on the server, the server knows the video playback rate before the starting the transfer.

The delay-based congestion control is used in order to utilize the network bandwidth effectively. The delay-based congestion control of TCP Stream is based on TCP Vegas [18]. The delay-based congestion control starts to operate when the loss window size equals to the $\lceil Rate \cdot RTT / MSS \rceil$. In a delay-based congestion control, the congestion status of the network is estimated based on the time taken for one round-trip, and then the delay window size is determined from the estimated value. The estimated value of the network congestion status, *Diff*, is given by:

$$Diff = \left(\frac{swnd}{\tau} - \frac{swnd}{RTT} \right) \tau \tag{2}$$

where τ is the minimum round-trip time. The estimated value, *Diff*, is set to zero when the expected transmission rate ($swnd / \tau$) and the actual transmission rate ($swnd / RTT$) are equal. Moreover, the estimated value becomes large as the actual transmission rate degrades. It has been shown that the network congestion status is so bad that this estimation value is large. Based on this estimated value, we determine the delay window size according to the following equation:

$$dwnd \leftarrow \begin{cases} dwnd + 1 & (if\,Diff < \alpha) \\ dwnd - 1 & (if\,Diff > \beta) \\ dwnd & (\text{otherwise}) \end{cases} \tag{3}$$

where α and β are control parameters that determine the number of packet send to the network. In TCP Stream, a maximum of $\lceil Rate \cdot RTT / MSS \rceil$ packets are sent to a network by the loss-based congestion control during the time taken for one round-trip time. The control parameters α and β are determined as follows, based on the configuration value of the control parameter of TCP Vegas:

$$\begin{aligned} \alpha = &\quad 2 + \lceil Rate \cdot RTT/MSS \rceil \\ \beta = &\quad 4 + \lceil Rate \cdot RTT/MSS \rceil \end{aligned} \tag{4}$$

Table 1 summarizes the definition of symbols used in this section.

Table 1. Definition of symbols

lwnd	loss window size
dwnd	delay window size
swnd	send-out window size
RTT	round-trip time
τ	minimum round-trip time
Diff	estimated value for the network congestion status
Rate	video playback rate
MSS	maximum segment size
α	control parameter for TCP Stream
β	control parameter for TCP Stream

5. Performance Evaluation

In this section, we evaluate the performance of TCP Stream using simulation experiments and the implemented system in an experiment network environment. Here, we show that TCP Stream solves the video streaming problems described previously and its effectiveness.

5.1. Simulation

Using ns-2 simulator [19], we ran simulation experiments at the packet level to confirm the effectiveness of TCP Stream. Figure 2 shows the network model for the simulation experiments, in which 10 TCP Stream/NewReno connections for video streaming, TCP connections for FTP, and one UDP flow share the single bottleneck link, which has a capacity of 100 [Mbit/s] and a delay of 40 [ms]. The arrival of UDP packets follows a Poisson process, with an average arrival rate of 30 [Mbit/s]. The links between the sender/receiver and the router have a capacity of 100 [Mbit/s]. We set a random value between 5 [s] and 15 [s] to the delay of the links between the sender/receiver and the router in order to prevent the synchronization of TCP connections.

In our simulation experiments, the UDP flow begins to transfer data when the simulation starts, and the TCP NewReno connection(s) begin transferring data at random time between 0 [s] and 10 [s] to create background traffic and competing traffic on the network. After 40 [s], either TCP Stream or TCP NewReno video streaming connections start to transfer every second. The packet size is 1500 [Byte]. The simulation finishes when the transfer of all video sequences is completed. We ran 20 simulations and used the simulation results (excluding the first 50 [s] of the simulation time) to calculate performance metrics, such as average throughput and average packet loss probability. The video sequence is 270 [MByte] in size, the playback rate is 3.6 [Mbit/s], and the playback time is 600 [s], which is equivalent to a 1080p video on YouTube. The playout delay of video streaming at the receiver is 10 [s].

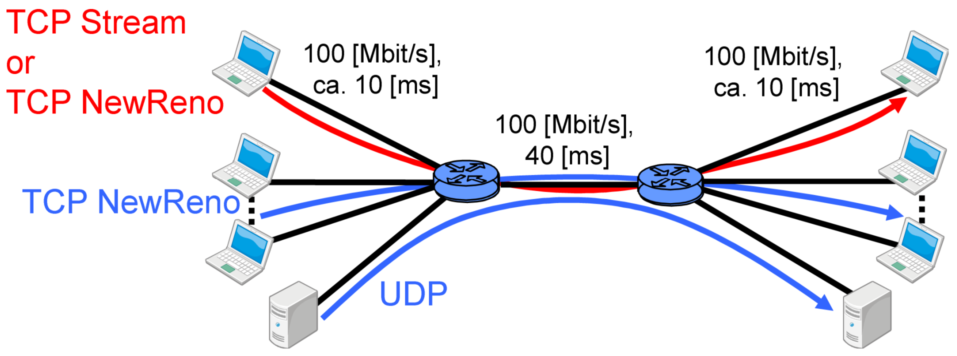
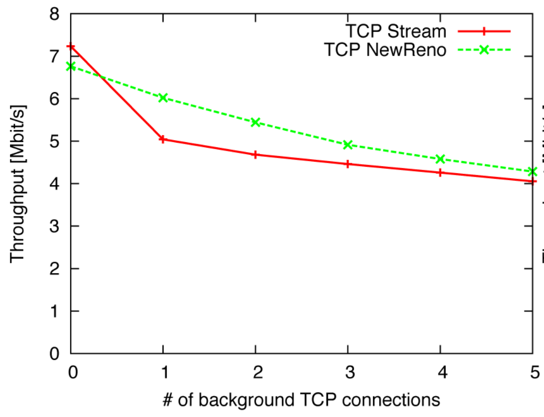
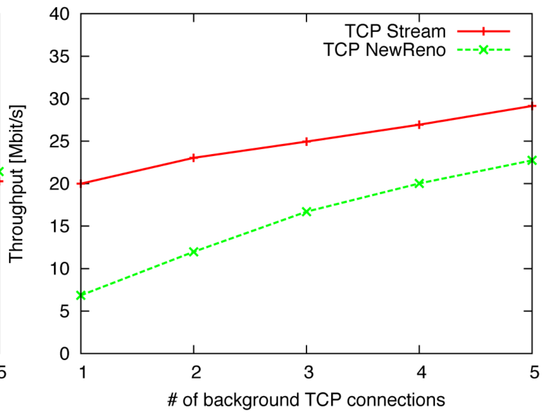


Figure 2. Simulation model

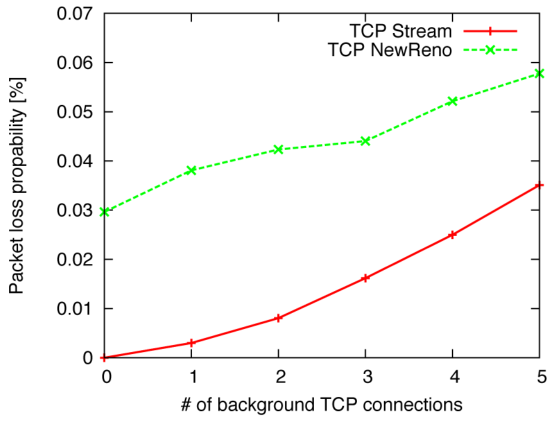
Figure 3 shows the simulation results as a function of the number of TCP connections. Figures 3(a)-(c) show: (a) the average throughput of TCP Stream/NewReno connections for video streaming, (b) the average packet loss probability in the network during the simulation experiments, and (c) the average total throughput of TCP NewReno connections for FTP. In Figure 3(a), we see that the TCP Stream transmission rate is quite high when a background TCP NewReno connection does not exist. This shows that TCP Stream utilizes the network opening bandwidth effectively by means of its delay-based congestion control. When the TCP NewReno connection(s) for FTP do exist, the TCP Stream transfer rate decreases, and TCP NewReno transfers data at a comparatively high rate. This degradation in the TCP Stream connection transfer rate is caused by losing the throughput competition with the TCP NewReno connections for FTP. Note that it is not necessary to transfer video data at a much higher rate than the video playback rate to maintain a continuous video playback.



(a) Average throughput of TCP Stream/NewReno for video streaming



(b) Average total throughput of TCP for FTP



(c) Average packet loss probability

Figure 3. Simulation results

Here, we focus on the total throughput of background traffic. From Figure 3(b), we see that the total throughput of background traffic when using the TCP Stream is higher than that when using TCP NewReno. This is because TCP NewReno transfers video data at a high rate, regardless of the video playback rate. As a result, the video streaming connections steal bandwidth from the other background traffic.

Figure 3(c) shows that the TCP Stream packet loss probability is lower than that of TCP NewReno. This is because TCP NewReno performs a greedy congestion control, and sends a lot of packets to the network. However, since the loss-based congestion control of TCP Stream has a maximum value, based on the video playback rate in the loss window size, TCP Stream dose not conduct a greedy congestion control. Consequently, when video sequences are transferred by TCP Stream, the network congestion status does not get worse, and the packet loss probability is lower than that of TCP NewReno.

From simulation results it turns out that, when TCP NewReno transfers the video sequences, it steals bandwidth from other network traffic, and so the packet loss probability is high. However, TCP Stream transfers video data without taking bandwidth from other network traffic unnecessarily.

5.2. Implemented System in the Experiment Network

We implement TCP Stream on the Linux kernel 2.6.32, and evaluate it in an experiment network environment. Figure 4 shows the experiment network environment. This network environment consists of a PC router in which DummyNet is installed, an end-host that transfers video sequences by TCP Stream/Cubic TCP, and an end-hosts that generates background traffic by Cubic TCP, and an end-host that receives packets from each end-host. Cubic TCP is the default TCP in Linux kernels and conducts loss-based congestion control [20]. All end-hosts and the PC router are connected by a 100 [Mbit/s] Ethernet connection. We configured the DummyNet setting so that the minimum round-trip time of all connections and a flow becomes 30 [ms]. Table 2 shows the specifications of the PCs of the experiment network environment. We change the network congestion status by changing background traffic over time as follows: from 0 [s] to 30 [s], there is no background traffic; from 30 [s] to 60 [s], there is one Cubic TCP connection. The playback rate of the video sequence to transfers is 3.6 [Mbit/s], which is equivalent to a 1080p video on YouTube.

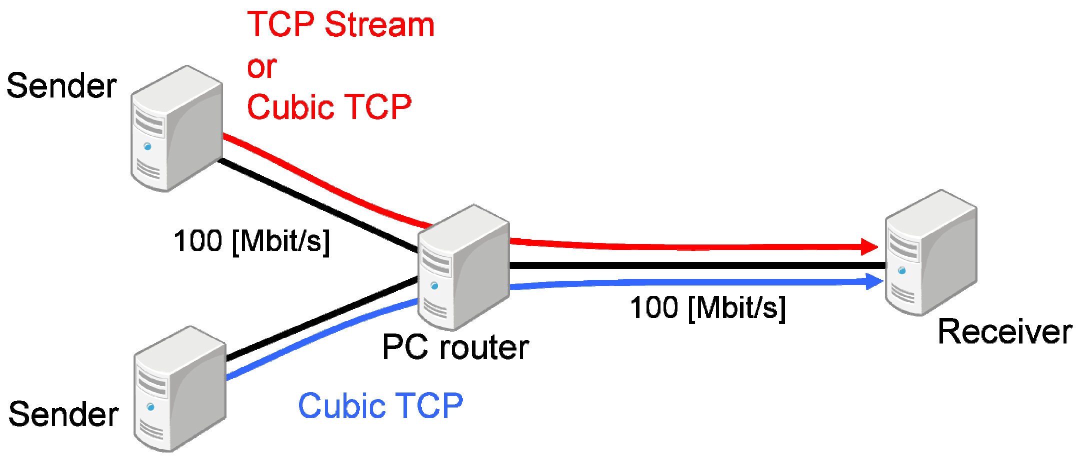
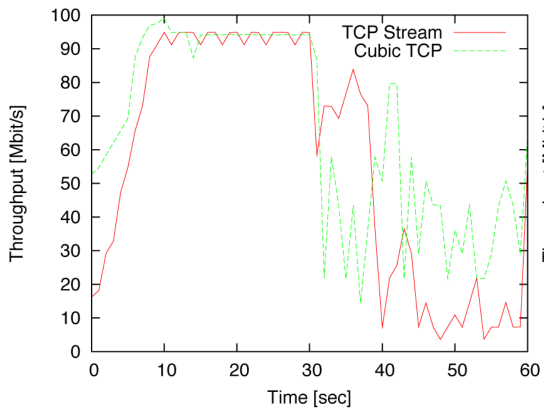


Figure 4. Experiment network

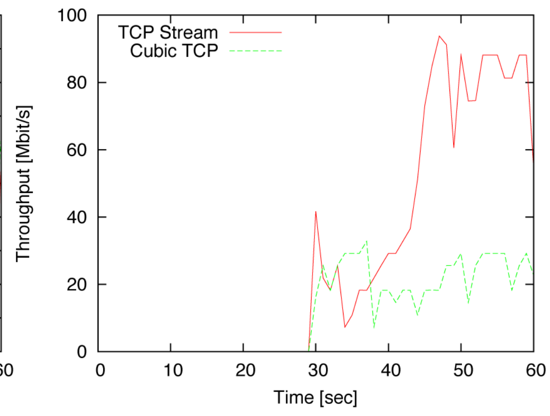
Table 2. PC specifications of the experiment network environment

	Sender	PC router	Receiver
CPU	Celeron 430: 1.80 [GHz]	Core2Quad: 3 [GHz]	Celeron 430: 1.80 [GHz]
Memory	2 [GByte]	8 [GByte]	2 [GByte]
OS	Linux (kernel 2.6.32)	FreeBSD 8.1	Linux (kernel 2.6.32)

Figure 5(a) and (b) show the changes in the throughput of both TCP Stream and Cubic TCP while streaming the video, and the changes in the total throughput of Cubic TCP for FTP, respectively. From Figure 5(a), one can find that when there is no background traffic, TCP Stream uses up the open bandwidth, similarly to the Cubic TCP and when there is background traffic, TCP Stream transmits data at the adjusted rate, unlike Cubic TCP. Moreover, from Figure 5(b), it turns out that the total throughput of background traffic with TCP Stream is larger than that with Cubic TCP. This is because that TCP Stream does not conduct the greedy congestion control. From these observations, we conclude that TCP Stream transfers video sequences appropriately.



(a) Changes in throughput of TCP Stream/Cubic TCP for video streaming



(b) Changes in total throughput of Cubic TCP for FTP

Figure 5. Experiment results

6. Conclusions and Future Works

In this paper, we have proposed a new transport-layer protocol for video streaming, called TCP Stream. TCP Stream performs window-based congestion control that combines two congestion controls: a loss-based congestion control that uses packet loss as an index of congestion, and a delay-based congestion control that uses a network delay as an index of congestion. We can use TCP Stream only by changing the sender-side, the services provider side, TCP and do not need to change the configuration of the operating system or modify the kernel at the receiver-side, the user side, for data transfer by TCP Stream.

As the results of our simulations, we have shown that TCP Stream can utilize open bandwidth when a network is not in a congestion state. Moreover, we have shown that when a network is in a congestion state, TCP Stream transmits data packets at an adjusted rate required for the video sequence, unlike TCP NewReno, and does not steal bandwidth from other network traffic. Furthermore, we have evaluated the implemented system in the experiment network environment, and have shown that TCP Stream operated appropriately.

As a future work, we would like to evaluate TCP Stream over the Internet. In addition, we would like to propose other useful mechanisms based on the network measurement results, and implement and evaluate these mechanisms over the Internet.

References

[1] "YouTube. " available at http://www.youtube.com/.

[2] "Dailymotion. " available at http://www.dailymotion.com/.

[3] "Veoh. " available at http://www.veoh.com/.

[4] S. Floyd, M.Handley, J.Padhye, and J.Widmer, (2000) "Equation-based congestion control for unicast applications," in *Proceedings of ACM SIGCOMM 2000*, pp. 43-56.

[5] L. Cai, X. Shen, J. Pan, and J. Mark, (2005) "Performance analysis of TCP-friendly AIMD algorithms for multimedia applications," *IEEE/ACM Transactions on Networking*, pp. 339-355.

[6] F. Yang, Q. Zhang, W. Zhu, and Y.-Q. Zhang, (2004) "End-to-end TCP-friendly streaming protocol and bit allocation for scalable video over wireless Internet," *IEEE Journal on Selected Areas in Communication*, pp. 777-790.

[7] S. L. Bangolae, A. P. Jayasumana, and V. Chandrasekar, (2000) "TCP-friendly congestion control mechanism for an UDP-based high speed radar application and characterization of fairness," in *Proceedings of the 8th International Conference on Communication Systems*, pp. 164-168.

[8] Y.Zhu, A.Velayutham, O.Oladeji, and R.Sivakumar, (2007) "Enhancing TCP for networks with guaranteed bandwidth services," *ACM Comuputer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 10, pp. 2788-2804.

[9] T.Tsugawa, N.Fujita, T.Hama, H.Shimonishi, and T.Murase, (2007) "TCP-AFEC: An adaptive FEC code control for end-to-end bandwidth guarantee," in *Proceedings of 16th International Packet Video Workshop (PV2007)* pp. 294-301.

[10] H. Hisamatsu, G. Hasegawa, and M. Murata, (2011) "Non bandwidth-intrusive video streaming over TCP," to be presented at *8th International Conference on Information Technology: New Generations (ITNG 2011)*.

[11] R. Kuschnig, I. Kofler, and H. Hellwagner, (2010) "Improving Internet video streaming performance by parallel TCP-based request-response streams," in *Proceedings of the 7th IEEE Conference on Consumer Communications and Networking Conference*, pp. 200-204.

[12] G. S. Rao, P. S. Kumar, K. D. Prasad, M. Supraja, G. V. Kumar, V. S. V. S. Murthy, M. P. Kumar, D. Rajesh, S. Lavanya, P. A. Deepthi, and C. P. Satish, (2010) "Architecture for efficiently streaming stored video using TCP," *International Journal of Computer Science and Network Security*, vol. 10, pp. 154-163.

[13] B. Libæk and O. Kure, (2009) "Protecting scalable video flows from congestion loss," in *Proceedings of the 2009 Fifth International Conference on Networking and Services*, pp. 228-234.

[14] H. Schulzrinne, A. Rao, and R. Lanphier, (1998) "Real time streaming protocol (RTSP)," *Request for Comments (RFC) 2326*.

[15] "Real-time messaging protocol (RTMP) specification." available at http://www.adobe.com/devnet/rtmp.html.

[16] J.Mo, R.J.La, V.Anantharam, and J.Walrand, (1999) "Analysis and comparison of TCP Reno and Vegas," in *Proceedings of IEEE INFOCOM'99*, vol. 3, pp. 1556-1563.

[17] K. Tan, J. Song, and Q. Zhang, (2006) "A compound TCP approach for high-speed and long distance networks," in *Proceedings of IEEE INFOCOM 2006*, pp. 1-12.

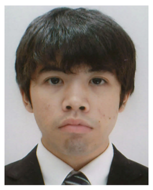
[18] M.Gerla, Y.Sanadidi, R.Wang, A.Zanella, C.Casetti, and S.Mascolo, (1994) "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM 1994*, pp. 24-35.

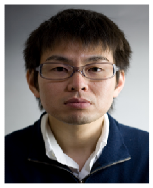
[19] "The network simulator – ns2." available at http://www.isi.edu/nsnam/ns/.

[20] S. Ha, I. Rhee, and L. Xu, (2008) "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, pp. 64-74.

Authors

Hiroki Oda received B.E and M.E. degrees from Osaka Electro-Communication University, Japan, in 2009 and 2011, respectively. He is currently a doctoral student at the Graduate School of Computer Science and Arts, Osaka Electro-Communication University. His research interests include network performance evaluation, TCP protocol design and evaluation. He is a student member of IEICE.





Hiroyuki Hisamatsu received M.E. and Ph.D. degrees from Osaka University, Japan, in 2003 and 2006, respectively. He is currently an associate professor of Department of Computer Science, Osaka Electro-Communication University. His research work is in the area of performance evaluation of TCP/IP networks. He is a member of IEEE and IEICE.

Hiroshi Noborio (M'87) was born in Osaka Prefecture, Japan, in 1958. He graduated in computer science from Shizuoka University in 1982 and received the Dr. Eng. degree (equivalent of Ph.D.) in mechanical engineering from Osaka University in 1987. From 1987 to 1988, he worked as Assistant Professor in the Department of Mechanical Engineering, Faculty of Engineering Science, Osaka University. He was a Humboldt Scholar and a visiting researcher in the Department of Electrical Engineering, Technical University of Munich. He is currently the Dean of Faculty of Information Science and Arts and also a Professor in the Department of Computer Science, Osaka Electro-Communication University. His research interests span robotics, computer graphics, virtual reality, and medical engineering. He has published over 100 scientific papers and papers of conference proceedings.