A Playout Buffer Efficient Multimedia Streaming Using Multiple TCP Connections

Young H. Jung*, Hyunghoon Lee*, In-Hwa Hong†, Yoonsik Choe*

School of Electrical & Electronic Engineering, Yonsei University, Seoul, Korea

{crosscom, hommeone, yschoe}@yonsei.ac.kr*

Digital Media Research Center, KETI, Seoul, Korea

hongih@keti.re.kr†

Abstract- **We designed a quality guaranteed multimedia streaming system especially when a server peer has insufficient bandwidth and has to perform both media streaming and content file transmission service simultaneously. To guarantee the bandwidth of a streaming session, the server uses multiple TCP connections in our proposed system. At this time, multiple TCP connections are dynamically created based on estimated play out buffer status of a streaming client. Simulation results show proposed algorithm can enhance streaming quality by successfully reducing the occurrence of playout buffer underrun.**

I. INTRODUCTION

[1]Owing to a great proliferation of the broadband Internet access, various multimedia services over the Internet such as IPTV broadcast and VOD (Video-On-Demand) prosper more and more these days. For these multimedia streaming services, UDP was thought as a preferable transport layer protocol than TCP in general. That is because as TCP uses retransmission and congestion control mechanism, TCP was regarded as not appropriate protocol for real-time application. However, the recent development of broadband Internet access and Internet infrastructure provides sufficient bandwidth and relatively short delay for each user and this leads successful multimedia service even with TCP transport [1]. Besides, various multimedia streaming services with TCP transport have been largely deployed more and more because TCP has many advantages over UDP such as firewall traversal issue, bandwidth fairness with other flows, and built-in reliable features [1].

In this paper, among various multimedia streaming service scenarios based upon TCP transport, the case of server with insufficient *first-mile* bandwidth is mainly addressed. (Through this paper, the term of first-mile means the link from a server to the next first hop in uplink direction.) For the example of this concerned environment, we can choose the P2P (peer-to-peer) network with broadband Internet access and mobile ad-hoc network. In our service model, a multimedia server with insufficient first-mile bandwidth tries to deliver multimedia content to other peers via both file transmission and media streaming service. During this process, the server uses FTP/TCP protocols for file transmission as usual and HTTP/TCP for multimedia streaming. Then, if both file transmission and media streaming service take place simultaneously, it is natural that each TCP flow fairly shares limited uplink bandwidth.

For a TCP flow fairly shares channel bandwidth with other TCP flows, the bandwidth of a media streaming flow can be shrunk accordingly as other TCP flows are created. If the bandwidth for a media streaming shrank below encoding rate of media content, then it can cause PoB (Play-out-Buffer) underrun which requires rebuffering time for a client. Furthermore this rebuffering eventually results in unwilling playback pause. As recent many researches such as [2][3] remarked, this unwilling playback pause due to rebuffering usually perceived as a severe quality degradation of streaming service. Thus, the streaming bandwidth shrinking due to fair-share characteristics of TCP can be directly connected to QoS (Quality-of-Service) degradation issue in this case.

In this paper, we proposed DMTS (Dynamic Multiple TCP connections for Streaming) algorithm to guarantee streaming quality when a server performs both media streaming service and file transmission service with limited first-mile bandwidth. To achieve this, multiple auxiliary TCP connections are created in order to guarantee the bandwidth of a streaming session when bandwidth shrinking is detected due to bandwidth fair-share among TCP flows. During this process, the creation of auxiliary TCP connections is controlled by the server based upon the PoB (play-out-buffer) status of a streaming client to minimize occurrence of possible buffer underrun. More specifically, the server gets feedback information from the client on current PoB status and measured packet arrival rate and then with these data, estimates the PoB status of near future. Based on this PoB estimation, if needed, server creates more auxiliary TCP connections to the streaming client. This scheme which uses multiple TCP connections is similar to the algorithm of [6]. In [6], fixed number of multiple TCP connections was used to prevent short-term bandwidth fluctuation during the streaming but we used multiple TCP connections to reserve enough bandwidth for media streaming and minimize the occurrence of PoB underrun in the view of long-term fair-share characteristics among TCP flows. And also due to dynamic control of the number of TCP connection, proposed DMTS algorithm utilizes system resource (e.g. TCP socket) effectively.

Through the simulation result, it is shown that proposed DMTS algorithm enhances the quality of multimedia streaming especially when media streaming and file transmission take place simultaneously from the server with limited first-mile bandwidth. That is, proposed system shows less frequent occurrence of PoB underrun and rebuffering rather than conventional system with one TCP connection or the system in [6] which has fixed number of multiple TCP connections. Thus, proposed DMTS algorithm can be effectively

[1] This work was supported by KMoCIE(Korean Ministry of Commerce, Industry and Energy) and KETI (Korean Electronics Technology Institute) under project number 2007-8-1213

applied to the multimedia content sharing service in P2P network or mobile ad-hoc network to enhance streaming quality without the network socket resource in a server system overused.

We present the major ideas in our paper as follows. In section II, we addressed related existing researches with concerned problematic service scenario. In section III, we cover the network and service environments, and some assumptions. In section IV, we show proposed whole DMTS algorithm which includes both of server and client roles. In section V, we present simulation result from network simulator. Last but not least, we address conclusion and future research interest in section VI.

II. Related Works

In [4], Mehra and Zakhor suggested BWSS (receiver-driven bandwidth sharing systems) for the solution to the case which is opposite to the one we have focused on. That is, in the case that a client has insufficient last-mile bandwidth, they solved the problem of bandwidth guarantee for multimedia streaming caused by TCP fair-share between streaming flow and other flows to the client. To achieve this, client regulates window size of each acknowledgement packet for whole receiving TCP flows to guarantee the bandwidth of streaming flow. Although BWSS shows good performance on guaranteeing streaming bandwidth in the case that client's last-mile bandwidth is not enough, it is not applicable to our concerned case that a server has limited first-mile bandwidth.

Gürses et al. adapted SFD (Selective Frame Discard) algorithm to streaming with TCP transport in which a server has limited bandwidth [5]. In this case, to achieve best quality of streaming service, the server measures available streaming bandwidth with the help of modified TCP stack and it controls streaming rate below the available rate by dropping chosen frames which can trigger minimum distortion. If the SFD algorithm applied to our concerned service scenario, the more file transmission flows increase, the more streaming clients suffer quality degradation by intentional frame drop.

Nguyen and Cheung suggested the use of multiple TCP connections (MultiTCP) for one streaming session in [6] to cope with short-term fluctuation of streaming bandwidth. In a streaming server, MultiTCP control unit which is located in the middle of TCP stack and application opens multiple TCP connections. Using these multiple TCP connections, congestion window is reduced less than at the case of one TCP connection during the congestion control stage; this contributes to decrease short-term bandwidth fluctuation. Although MultiTCP can prevent short-term bandwidth fluctuation, it cannot properly handle the long-term decrease in available bandwidth which has been caused by fair-share characteristics of TCP. That is because it uses fixed number of TCP connections for a streaming session regardless of currently available bandwidth.

III. Network And Service Environment

The service architecture and network model is depicted in Fig. 1. As shown in the figure, each peer which spreads out in the Internet can be both of server or client for multimedia contents sharing service via media streaming and file transmission. Because each peer connects to the service network through using broadband Internet access or wireless Internet access, it has relatively smaller bandwidth rather than backbone network. Especially, more often than not, uplink bandwidth of each server is also much smaller than its downlink bandwidth due to bandwidth asymmetry characteristics of commercial broadband Internet access such as A(V)DSL and wireless Internet such as WCDMA, mobile WiMAX.

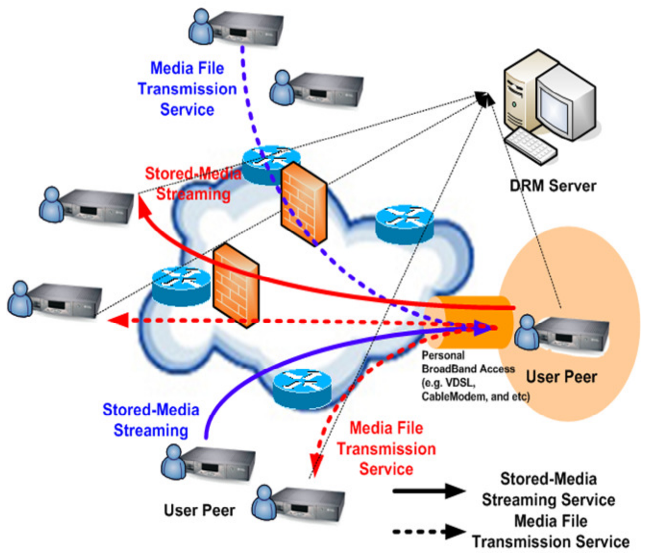


Figure 1. Multimedia contents sharing service architecture.

With this network architecture, to cope with firewall traversal issues and achieve reliable transmission, TCP is just used for multimedia streaming for transport layer protocol. Also, media file transmission is performed with FTP/TCP protocols used in general.

In application layer, a streaming client sends feedback on its PoB status and measured data arrival rate periodically with configured interval. On the other hand, a streaming server sends out pre-stored CBR (constant bit rate) multimedia streaming data when a client requests. And to guarantee streaming quality, the server application analyzes and estimates the status of client PoB in the near future. Media streaming services and media file transmission services can take place simultaneously at the server by various requests from different clients. Then, because both media streaming and file transmission service use TCP for transport layer, each flow fairly shares limited uplink bandwidth as shown in Fig. 2.

In this paper, we assumed that streaming service gets higher priority than media file transmission service. In other words, we evaluated entire service quality based upon streaming service quality, not based upon file transmission. Also, streaming service quality is just measured by the count of PoB underrun occurrence rather than frame distortion because we have decided to achieve reliable and lossless transmission using TCP transport. The count of PoB underrun occurrence is closely related to subjective streaming quality as noticed in recent many researches such as [2] and [3].

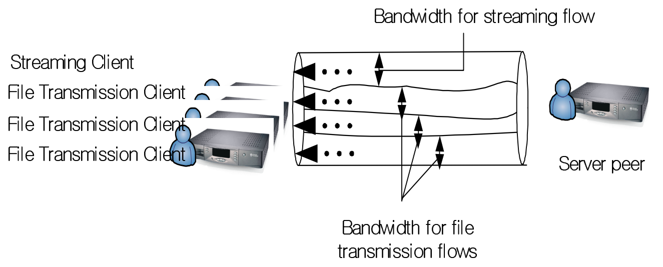


Figure 2. Example of bandwidth fair share among streaming flow and file transmission flows.

IV. Playout Buffer Efficient Dynamic Multi-TCP Algorithm

A. *Streaming Models*

Streaming model for proposed algorithm is described in Fig. 3. Let $A(t)$ denote the total arrived bytes at a client by time, t. Because $A(t)$ is dependent on network status, we just let this as a random process. And let $P(t)$ denote the total played bytes at the client by the time t. As usual, streaming client application uses pre-roll buffering to reduce the effect of delay variation from the network. We denoted this pre-roll buffering time as τ . And CBR playback rate which is same with encoding rate of media content is denoted as μ. Then, $P(t)$ can be expressed as (1).

$$P(t) = \begin{Bmatrix} \mu(t-\tau), & t > \tau \\ 0, & t < \tau \end{Bmatrix} \tag{1}$$

Also let $B(t)$ as the total remain bytes at the client PoB by the time t and then we can denote $B(t)$ with (2).

$$B(t) = A(t) - P(t) \tag{2}$$

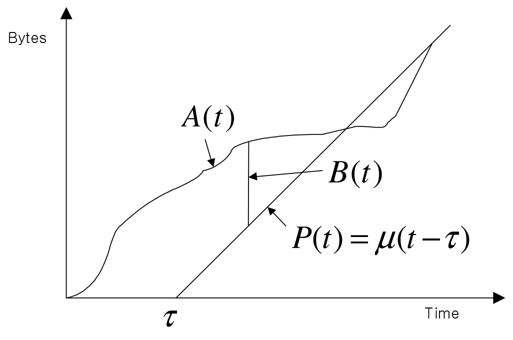


Figure 3. Streaming model

B. *Playout buffer estimation at streaming*

To guarantee streaming bandwidth while performing both media streaming and file transmission service simultaneously over limited uplink bandwidth of a server, a server application should estimate the status of client PoB in the near future during the service. That is, $B(t_{curr} + \Delta t)$ should be estimated in the server at time t_{curr} .

Although $A(t)$, $(t \le t_{curr})$ is known to the server using periodic feedback information from the client, $A(t_{curr} + \Delta t)$ is not deterministic because of network dynamics. Therefore, we used window based linear regression to estimate $A(t_{curr} + \Delta t)$ approximately. That is, $A(t)$ can be modeled with the form of (3).

$$A(t) = \alpha + \beta \cdot t + \varepsilon \tag{3}$$

Where α and β are regression parameters, and ε means zero mean Gaussian random variable. With previous w samples, the server module estimates regression parameters which yield least mean square error as follows. [7]

$$\beta = \frac{\sum_{w} t_i A_i - \bar{t} \sum_{w} A_i}{\sum_{w} t_i^2 - w\bar{t}^2} \qquad \alpha = \bar{A} - \beta\bar{t} \tag{4}$$

Where $\bar{A}$ and $\bar{t}$ means average arrived bytes and average sample time among previous w samples at t_{curr} .

C. *Guard interval for new TCP connections*

As known well, if a new TCP connection created, then TCP connection probes available network bandwidth by using congestion control algorithm. Then, if there is no other network change such as the creation of another TCP flows and etc, the bandwidth of TCP connection would be stabilized to some level after a few second. We call this period as a guard interval. If a server dynamically has created one or more new TCP connections, then it should stop creating new TCP connections during this guard interval. This is because a guard interval itself means expected time until newly created TCP connection shows some effect. Therefore, guard interval can prevent excessive use of socket resource of the server system.

Since guard interval depends on network condition and characteristics, it is better to choose appropriate value by real-time estimation rather than pre-fixed one. To determine guard interval, the server module uses the measured arrival rate which is embedded in feedback information from the streaming client. Using these data, the server module calculates moving average of arrival rate from the beginning of the streaming session. By definition, guard interval can be written with (5).

$$t_{guard} = t_{stab} - t_{init} \tag{5}$$

Where, t_{init} means the time at which first feedback information received and t_{stab} means the time at which (6) would be satisfied.

$$\begin{aligned} &R_{avg}(t) = m \times R_{avg}(t-1) + n \times R_{measured}(t) \\ &t = t_{stab}, \quad if \left| R_{avg}(t) - R_{measured}(t) \right| \le R_{avg}(t) * \sigma \end{aligned} \tag{6}$$

Where, $R_{avg}(t)$ means the moving average of arrival rate at the client and $R_{measured}(t)$ means measured arrival rate at the client. Definitely, $R_{measured}(t)$ would be reported to the server via feedback information periodically. In (6), m and n are moving average parameters and σ is the parameter for affordable bandwidth variation.

D. *Dynamic MultiTCP streaming Algorithm*

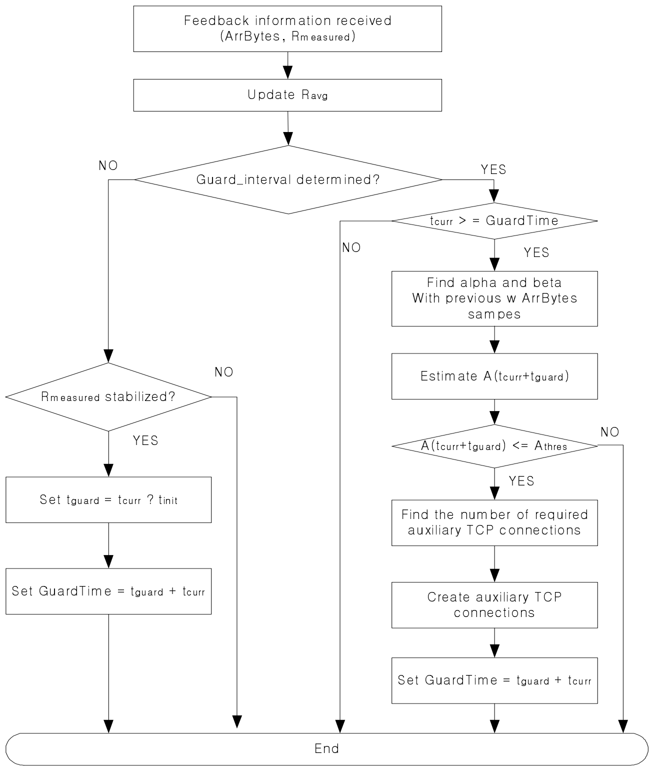

Feedback information received (ArrBytes, Rmeasured)
Update Ravg
Guard_interval determined?
NO
YES
tcurr > = GuardTime
NO
YES
Find alpha and beta With previous w ArrBytes sampes
Estimate A(tcurr+tguard)
A(tcurr+tguard) <= Athres
NO
YES
Find the number of required auxiliary TCP connections
Create auxiliary TCP connections
Set GuardTime = tguard + tcurr
Rmeasured stabilized?
NO
YES
Set tguard = tcurr ? tinit
Set GuardTime = tguard + tcurr
End

Figure 4. Flow chart of server process

Fig. 4 shows entire procedure of proposed DMTS (Dynamic MultiTCP streaming) algorithm at the server when it received feedback information from the client. At the beginning of streaming service, the server firstly determines guard interval for one TCP connection using measured arrival rate from the client. And then, whenever the server receives feedback information from the client, it anticipates the PoB status of near future using linear regression model of arrived bytes and deterministic playback model. At this time, we use t_{guard} for the estimation interval of arrived bytes (Δt) based upon the definition of guard interval. If estimated PoB status falls below the configured threshold, then the server module creates auxiliary TCP connections. To reduce computational complexity of the server application, the number of auxiliary TCP connections that will be newly created is determined by following simple proportional equation.

$$\begin{aligned} l + k_{curr} : k_{curr} &= R_{tot} : R_{avg} \\ l + k_{curr} + k_{new+} : k_{curr} + k_{new+} &= R_{tot} : \mu \end{aligned} \tag{7}$$

In (7), l stands for the number of total file transmission flows, k_{curr} for the number of currently opened TCP connections for a streaming session, k_{new+} for the number of auxiliary connections which are newly required, and R_{tot} means total bandwidth of server's uplink channel. We assumed that R_{tot} is pre-configured to the server application. Then, proportional equation of (7) yields k_{new+} as follows,

$$k_{new+} = \left\lfloor \frac{(\mu - R_{avg}) \cdot R_{tot}}{(R_{tot} - \mu) \cdot R_{avg}} \times k_{curr} \right\rfloor \tag{8}$$

Once a server created new auxiliary TCP connections, it would not perform PoB estimation and threshold check algorithm any more during the guard interval. This is due to prevent unnecessary overuse of TCP connections as explained in section *C*.

V. SIMULATION RESULTS

In this section, we verify the quality enhancement of proposed DMTS algorithm by simulation results. For simulation, we used network simulator, NS-2 [8] and EvalVid [9] module with some changes to support TCP streaming. We used MPEG-4 compressed bitstream which is encoded with CBR 512kbps, 30 frames/sec. Pre-roll time of streaming client is set to 5 sec and feedback period of the client is set to 300 msec. Through various experiments, we choose $m = 0.8$, $n = 0.2$ and $\sigma = 0.05$ which show good overall estimation result for average arrival rate. And also linear regression window size w is fixed as 20 samples to provide reliable estimation and ensure low complexity. To simulate simultaneous file transmission service during the streaming service period, we used general FTP module in NS-2. Other parameters that related with network topology are show in Fig. 5.

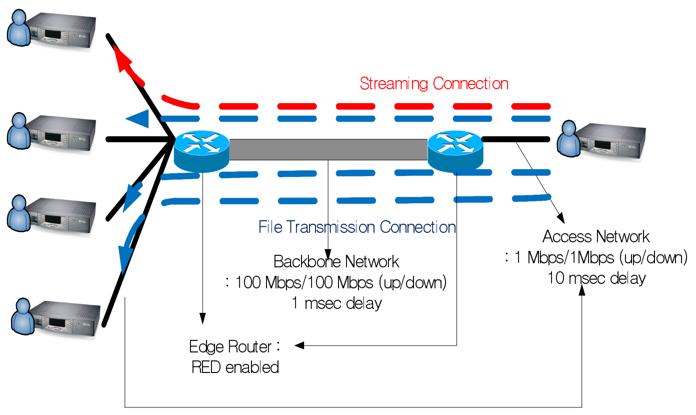


Figure 5. Simulation Network Topology.

With these parameters and network topology, we build two service scenarios to show quality enhancement of proposed algorithm over various situations.

- ■ *Scenario 1*:
  - Multimedia streaming starts at time 1 sec
  - File transmissions start at time 15, 25, 35, 40 sec respectively
- ■ *Scenario 2*:
  - File transmissions start at time 1 , 10 sec
  - Multimedia streaming starts at time 30 sec

- Other file transmissions start at time 45, 60 sec respectively

For comparison, we simulated above scenarios with 3 different types of system. That is, systems with proposed algorithm, traditional one TCP connection for a streaming, and fixed number of MultiTCP connections [6] are used respectively. Fig. 6 and Table I show the simulation result on scenario1 among three systems. As shown in Fig. 6, after file transmission services started (after about 30 sec in Fig. 6), the stocked data in the PoB of a streaming client decreases. At that time, because the streaming server with the proposed DMTS creates more auxiliary TCP connections for streaming session, it can quickly recover enough data to play out, whereas the server with other systems suffers sluggish playback due to PoB underrun.

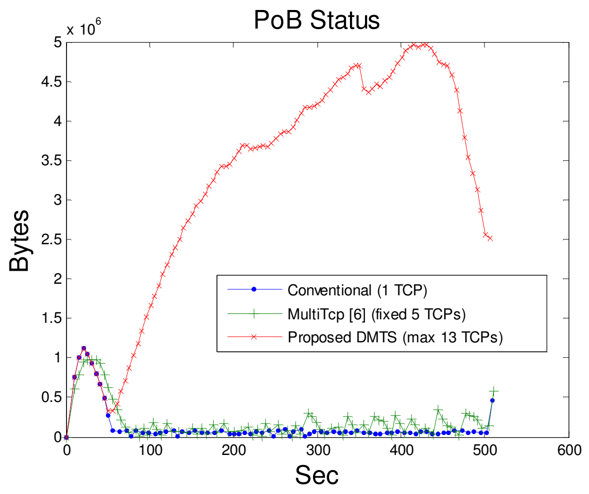


Figure 6. PoB status result (service scenario 1)

TABLE I
Simulation result on scenario 1

	Number of TCPs for one streaming session	Rebuffering occurrence
Conventional	1	**60**
MultiTCP [6]	5	**31**
Proposed DMTS	13(max)	**0**

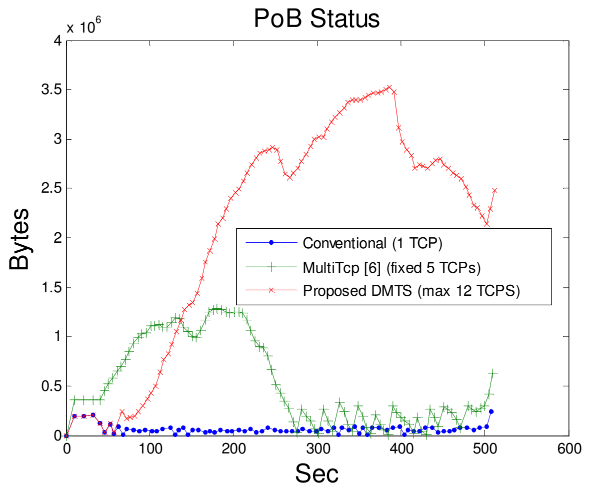
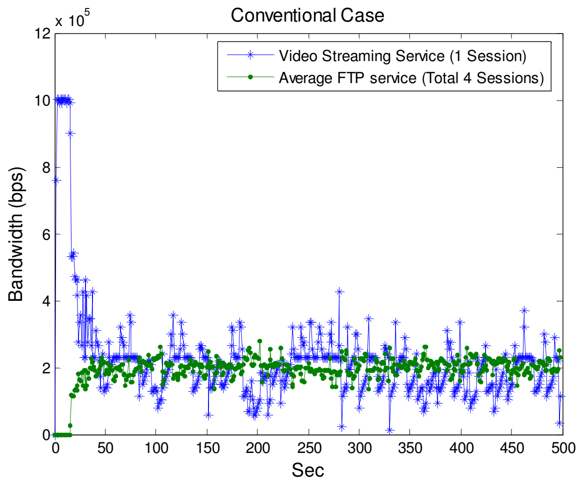


Figure 7. PoB status result (service scenario 2)

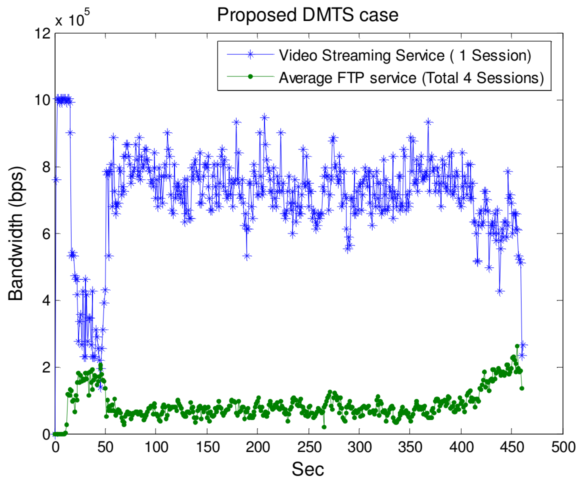
TABLE II
Simulation result on scenario 2

	Number of TCPs for one streaming session	Rebuffering occurrence
Conventional	1	**59**
MultiTCP [6]	5	**10**
Proposed DMTS	12(max)	**2**

Similar with the result of scenario 1, we were able to demonstrate the quality enhancement of proposed DMTS algorithm also in scenario 2 with Fig. 7 and Table II. In this case, because MultiTCP [6] uses fixed multiple TCP connections for a streaming session from the session beginning, it shows better performance than proposed DMTS during the first 150 seconds interval. However after this interval, proposed DMTS shows much better performance than MultiTCP [6] because it keeps creating more auxiliary TCP connections until enough streaming bandwidth reserved.



(a) Conventional system case (1 TCP for streaming session)



(b) Proposed DMTS case (D-MultiTCPs for streaming session)

Figure 8. Bandwidth variation in service scenario 1
(File transmission service vs. media streaming)

In Fig. 8, the average file transmission rate of 4 FTP services and streaming rate in the scenario 1 is presented. As shown in Fig.8 (a), the streaming bandwidth of the server with conventional 1 TCP connection is almost analogous to the average bandwidth of 4 FTP services due to fair-share characteristics of TCP as addressed in section II. However, the server with proposed DMTS can guarantee enough bandwidth for a media streaming as depicted in Fig. 8 (b).

VI. Conclusions

In this paper, we have designed DMTS algorithm to enhance streaming service quality when a media server has insufficient uplink bandwidth and has to serve both media streaming and file transmission service. To guarantee streaming service bandwidth, we used multiple TCP connections for one streaming session. And multiple TCP connections are created dynamically based upon estimated PoB status of a client. Thus, we can successfully prevent the occurrence of rebuffering during the streaming under the assumed service scenarios. This proposed algorithm can be effectively adopted to streaming service environment with P2P network or mobile ad-hoc network in which peer user has insufficient uplink bandwidth. And our future research interest will be in this service environment, such as a quality guaranteed bi-directional streaming over asymmetric bandwidth path.

References

[1] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study", In *Proc. of ACM Multimedia*, Oct. 2004, pp. 908-915.

[2] A. C. Dalal, and E. Perry, "A New Architecture for Measuring and Assessing Streaming Media Quality", In *PAM workshop 2003*, Apr. 2003, pp. 223-231

[3] N. Terada, E. Kawai, and H. Sunahara, "Extracting Client-Side Streaming QoS Information from Server Logs", In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing,* Aug.2005, pp.621-624.

[4] T. Nguyen and S. S. Cheung, "Multimedia Streaming Using Multiple TCP Connections", in *IPCCC 2005*, Apr. 2005, pp 215- 223K

[5] E. Gürses, G. B. Akar and N. Akar, "A simple and effective mechanism for stored video streaming with TCP transport and server-side adaptive frame discard", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Volume 48 , Issue 4, Jul. 2005, pp 489-501

[6] P. Mehra and A. Zakhor, "TCP-Based Video Streaming Using Receiver Driven Bandwidth Sharing", in *International Packet Video Workshop 2003*.

[7] S.M. Ross, *An Introduction to Probability and Statistics for Engineers and Scientists,* 2nd ed., A Harcourt Academic Press, 2000, pp.325-403

[8] ns-2 Network Simulator, available from http://nsnam.isi.edu/nsnam/index.php/Main_Page, (2007).

[9] C. Ke, C. Lin, C. Shieh, W. Hwang, "A Novel Realistic Simulation Tool for Video Transmission over Wireless Network", *IEEE International Conference on SUTC2006*, June 5-7, 2006, Taichung, Taiwan.