I.J. Computer Network and Information Security, 2015, 1, 24-30
Published Online December 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2015.01.04

Mean Response Time Approximation for HTTP Transactions over Transport Protocols

Y. –J. Lee
Department of Technology Education, Korea National University of Education, South Korea
Email: lyj@knue.ac.kr

Abstract—This paper addresses mean response time that end-users experience when using the Internet. HTTP (Hyper Text Transfer Protocol) is a widely used transfer protocol to retrieve web objects in the Internet. Generally, HTTP uses TCP (Transmission Control Protocol) in a transport layer. But it is known that HTTP interacts with TCP inefficiently. As an example of such inefficiencies, HTTP does not require TCP to deliver the rigid order, which may cause head-of-line blocking. As another transport layer protocol, SCTP (Stream Control Transmission Protocol) has attractive features such as multi-streaming and multi-homing unlike TCP. Within an SCTP association, multi-streaming allows for independent delivery among streams, thus can avoid the head-of-line blocking. In addition, SCTP provides very large number of streams; therefore, it can transfer multiple objects more efficiently than the typical HTTP/1.1 over TCP which limits the number of pipelines. Mean response time is one of the main measures that end users using Internet concern. This paper presents the simple analytical model and algorithm to find the mean response time for HTTP over SCTP including the previous HTTP over TCP. Some computational experiences show that the proposed model and algorithm are well approximated to the real environment. Also, it is shown that mean response time for HTTP over SCTP can be less than that for HTTP over TCP.

Index Terms—Mean response time, HTTP over TCP, HTTP over SCTP.

I. Introduction

Mean response time is one of the important measures to evaluate the performance of the Internet service. HTTP is a main application protocol to offer the Internet service. Since HTTP requires a reliable transfer, it uses connection oriented protocol like TCP and SCTP in the transport layer.

HTTP 1.0 does not provide the means to request the multiple objects, thus, we must establish a new TCP connection for retrieving each object from the web server. Since HTTP 1.0 requires two extra RTTs (round trip times) in setting up a new TCP connection between the web client and the web server, it is particularly inefficient [1].

HTTP 1.1 aims to reduce the extra setup time with persistent connections. It allows the web client to retrieve all objects from the web server by using pipelining. But, the numbers of objects which pipeline can handle simultaneously are affected and limited by the processing power of web server [2].

Even though any enhanced version of HTTP is used, there is a mismatch between the requirements of HTTP and the functionality by TCP. When multiple embedded objects are transmitted by using HTTP, TCP wants that each object should be reliably transferred. However, ordered delivery of these objects is not a requirement of HTTP. Instead, HTTP user wants the perceived latency to be reduced. In fact, most users are only concerned about fast response time.

TCP offers only a single stream of data and requires rigid ordered delivery. While this feature is desirable for delivery of a file or record, it causes additional delay when message loss or sequence error occurs in the network. When loss or error happens, TCP should delay delivery until the sequencing is corrected, either by receipt of an out-of-sequence data, or by retransmission of a lost data.

For a number of applications, this strict sequence preservation is not truly necessary. Web application needs not to maintain sequence between the presentations of objects. In some cases, web application may present parts of a single object out of sequence. Our goal is to deliver all objects as soon as possible. However, the ability to deliver objects out of sequence can achieve better performance. The reason is why parts of the web page can be displayed before waiting the entire objects. This situation is called the head-of-line blocking.

SCTP (Stream Control Transfer Protocol) [3] is a newly suggested message oriented transport layer protocol. SCTP provides a reliable full-duplex connection, called an association, and employs the control congestion mechanism like TCP. However, SCTP provides enhanced delivery options which TCP does not offer. SCTP can divide data into multiple streams and deliver independently by using its own multi-streaming function.

By this function, SCTP can limit message loss in any of the streams within that stream only.

We can use multi-streaming for independent delivery among streams within an SCTP association, thus reduce the head-of-line blocking situation [4]. Furthermore, within a single SCTP association, we can create so many unidirectional streams by either end for simultaneous data transfer. Therefore, if one object is lost during the transfer, the other objects can be transmitted while the lost object is retransferred. This will result in a better response time of end-users through simultaneous retrieval of the multiple objects [5]. In contrast, the numbers of pipelines for HTTP 1.1 over TCP are limited largely by the capability of web server.

Now, we investigate the TCP model related with the HTTP. TCP model [6] considered the TCP bulk data in a steady state. However, average transfer size of each TCP connections is known 8-12 KB [7]. Thus Most TCP connections carry very short data in Internet. Small size of web object affects the performance of web application by startup effects such as connection establishment and slow start. Cardwell et al. [8] extended the previous steady-state model, but, does not consider the effects by the slow start after timeout. Jiong et al. [9] improved model [8] by considering the influence of slow start period after retransmission timeout, however, does not consider multiple packet losses.

Particularly, the above researches focus on the micro behavior only and do not consider the head-of-line blocking time and multiple packet losses. But, most end-users are not interested in the theoretical TCP latency, but the overall response time.

Chang et al. [10] conducted the research about the performance of File Transfer Protocol (FTP) over SCTP, and the performance of Session Initiated Protocol (SIP) over SCTP was analyzed in [11]. A simple closed-form formula to estimate the HTTP latency over FAST TCP, taking into account the network parameters such as packet size, link capacity, and propagation delay was presented in [12]. Eklund et al. [13] developed a prediction model for the transfer times of SCTP messages during slow start. However, the mean response time approximation model for HTTP over SCTP in various situations has not yet been presented.

Lee et al. [14] presented mean response time estimation model for HTTP over SCTP in wireless environment and Lee [15] proposed more detailed model by extending the model [14]. However, because both models focus on the behavior of congestion control mechanism, they did not provides mean response times according to several HTTP versions and transaction processing methods.

Based on [14] and [15], this paper presents the approximation model for mean response time of several HTTP transactions methods over TCP and SCTP in transport layer when the packet loss occurs only during slow start phase due to small bandwidth.

We begin by presenting modeling and algorithm of HTTP transactions in the next section. In section 3, we describe HTTP applications to the model and Section 4 describes performance evaluation. We present our conclusions in section 5.

II. Modeling and Algorithm for Mean Response Time of HTTP Transactions

A. *Modeing for mean response time*

We first assume that each size of objects is identically distributed and the packet loss probability is given by any constant value. We send the HTTP request for one object and receive an acknowledgement from the server. Fig. 1 represents such a procedure. In the Fig. 1, *RTT* shows round trip time between the client and the server. S_1, S_2,.., S_a shows the slow start time until the first, second,, ath packet loss occurs respectively. DT_1, DT_2,.., DT_a represent the data transfer time for part of object until the first, second,.., ath packet loss occurs respectively. In addition, HOB_1, HOB_2,., HOB_a represent the head-of-line blocking time that requires the waiting for the retransmission.

When the total number of packets in object (N) is N=O/b where O and b are the size of the object to be transferred (bits) and maximum segment size (bits), respectively. When the packet loss probability is p, the expected number of packet loss (a) is a=Np according to the binomial distribution.

In Fig. 1, total response time = *RTT* + object transfer time = $RTT + (S_1 + DT_1 + TR_1 + HOB_1) + \cdots + (S_a + DT_a + TR_a + HOB_a)$. Here, TR_i and DT_i (i=1,..,a) shows data retransmission time and data transfer time in relation to i^{th} packet loss respectively .

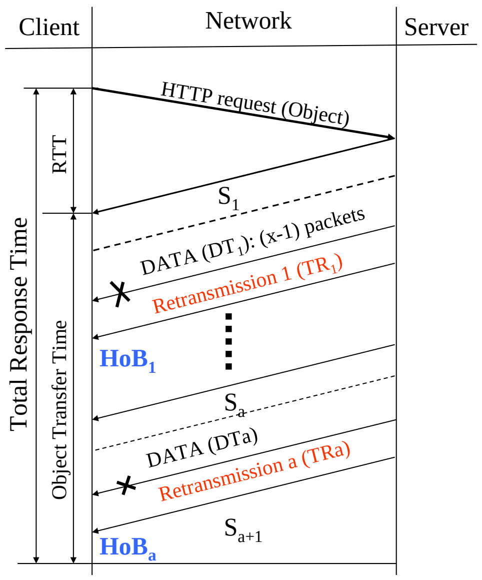

Fig. 1. HTTP transaction and total response time

Since *RTT* can be regard as constant value in the given environment, total response time varies according to the object transfer time. But, the sum of the data transfer time is $\Sigma_{k=1}^{a} DT_k = O/Q$ and can be treated as the constant. Here, Q represents transmission rates of the link from the server to the client (bps). Therefore, slow start time (S_i), retransmission time (TR_i) and head-of-line blocking time (HOB_i) in relation to i^{th} packet loss are affecting elements to the total response time. Without packet loss, we need not the additional slow-start time, retransmission time and head-of-line blocking time. Thus, in that case, only the slow start time and data transfer time are required to find out response time.

Now, we assume that packet loss occurs. First, in order to find the slow start time (S_1) until the first packet loss, we have to know the expected number of packets sent before the loss. We can determine the expected number of packets sent until a packet loss excluding the lost packet itself (x) by (1). Thus, it is natural that the expected number of packets sent until the first packet loss is (x-1).

$$x = \left\lceil p\sum_{k=1}^{N} k(1-p)^{k-1} \right\rceil \approx \left\lceil \frac{1-(1-p)^N}{p} + (1-p)^N \right\rceil \quad (1)$$

In general, slow start time (ST) for web object transfer is given by [16]

$$ST = c \times [RTT + \frac{b}{Q}] - (2^c - 1) \times \frac{b}{Q} \quad (2)$$

Here, c = min [V, W-1]. V and W represent the number of times the server would stall and the number of windows covering the object when the packet is not lost, respectively. V is given by (3). W is given by (4).

$$V = \left\lfloor \log_2(1 + \frac{RTT \times Q}{b}) \right\rfloor + 1 \quad (3)$$

$$W = \left\lceil \log_2(\frac{O}{b} + 1) \right\rceil \quad (4)$$

However, when the packet is lost, we should send the (x-1) packets before the loss, therefore, W is given by

$$W = \lceil \log_2(x - 1 + 1) \rceil \approx \lceil \log_2 x \rceil \quad (5)$$

By substituting these values into the above ST, we can obtain the slow start time until the first packet loss (S_1).

The retransmission time for one segment (TR_1) is $TR_1 = b/Q$.

Next, we compute the window size (y) covers the expected value of packet number when the loss occurs (x).

$$\begin{aligned} y &= \min\{W : 2^0 + 2^1 + \cdots + 2^{W-1} \geq x\} \\ &= \min\{W : 2^W - 1 \geq x\} \\ &= \min\{W : \log_2(x+1)\} = \lceil \log_2(x+1) \rceil \end{aligned} \quad (6)$$

Since we can infer that when the packet loss occurs, the half of number of packets in the window on the average are waiting for the retransmission for reordering, mean head-of-line blocking time due to the first packet loss can be given by

$$HOB_1 = \frac{yb}{2Q} \quad (7)$$

We can obtain the remaining values of S_2,.., S_{a+1} so on by using the same method. It is noted that S_i must be found for (i=1,.., a+1), but HOB_i, DT_i, and TR_i must be found for (i=1,..,a).

B. Algorithm for object transfer time

Based on the above model, we can formulate the entire procedure to find the object transfer time.

Algorithm 1. Pseudo-code for object transfer time

```
STEP 1:
① Set dt (data transfer time) = $O/Q$.
   Set st (slow start time) = 0.
   Set hob (head-of-line blocking time) = 0.
   Set rt (retransmission time) = 0.
② Compute the total number of packets in object, $N=O/b$
   and the expected number of packet loss, $a=Np$.
③ Set $i = 0$.
STEP 2:
① If $a = 0$ ($p = 0$), then proceed to ④ of STEP 2.
② Set $i = i + 1$.
   If $i = a+1$, Set $x = N$ and go to ④ of STEP 2.
③ Compute the expected number of packets sent until a
   packet loss by (1).
④ // slow start time computation phase
   Compute the slow start time ($S_i$) by using (2)
   Compute the number of times the server would stall
   by (3) and the number of windows that cover the object
   without packet loss by (5), respectively.
   Set $st = st + S_i$.
   If $a = 0$ or $i = a+1$ then proceed to STEP 4.
⑤ Compute the retransmission time ($TR_i$),
   Set $TR_i = b/Q$.
   Set $rt = rt + TR_i$.
⑥ Compute the window size($y$) covers the expected value
   of packet number when the loss occurs ($x$) and mean
   waiting time ($HOB_i$) by (6) and (7), respectively.
⑦ Set $hob = hob + HOB_i$.
STEP 3:
   If $i = a+1$, then proceed to STEP 4.
   Otherwise, Set $N = N - x$ and return to STEP 2.
STEP 4:
   Compute object transfer time ($OT$),
   $OT = dt + st + rt + hob$
```

In the first step, we introduce the following local variables; *dt*, *st*, *hob*, and *rt*. *dt* represents data transfer time, and is given by (object size)/(link transmission rate). *st* shows slow start time and is equal to 0 at the initial phase. *hob* shows the head-of-line blocking time and is also given by 0 at the initial phase. Finally, *rt* means the retransmission time, and is set to zero as an initial value.

We then compute the total number of packets included an object (N) is given by (object size / maximum segment size) and the expected number of packet loss (a) is computed by (the total number of packets * the packet loss ratio). We set the iteration count (i) to zero.

In the second step, we first check whether the expected number of packet loss is equal to zero. That is, we investigate if the packet loss ratio is equal to zero. If so, then we proceed to the slow start time computation phase, otherwise, we increase the iteration count (i) by one. If the increased iteration count is equal to (a+1), we set the expected number of packets sent until a packet loss excluding the lost packet itself (x) to the total number of packets included an object (N). Otherwise we compute the expected number of packets sent until a packet loss by (1).

In the slow start time computation phase, we compute the slow start time (S_i) by using (2) and compute the number of times the server would stall by (3) and the number of windows that cover the object without packet loss by (5), respectively. The total slow start time (*st*) is computed as the sum of the previous slow start time (*st*) and S_i obtained in this phase. If the computed number of packet loss (*a*) is equal to zero or the iteration count (*i*) is equal to *a*+1, we jump to the last step.

We compute the retransmission time (TR_i) using equation, TR_i = maximum segment size *(b)*/ transmission rates of the link (*Q*). And we compute the retransmission time (*rt*) by $rt = rt + TR_i$. Then we compute the window size(*y*) covers the expected value of packet number when the loss occurs (*x*) and mean waiting time (HOB_i) by (6) and (7), respectively. We accumulate the head-of-line blocking time (*hob*) by $hob = hob + HOB_i$.

In the third step, we investigate whether the iteration count (*i*) is equal to *a*+1, if so, we go to the last step. Otherwise, we decrease the number of packets to be sent by setting $N = N - x$ and return to the second step.

In the last step, we compute the total object transfer time (*OT*), by using equation, *OT* = data transfer time (*dt*)+ slow start time (*st*) + the retransmission time (*rt*) + the head-of-line blocking time (*hob*).

Algorithm 1 summarizes the above procedure and shows the pseudo-code.

III. Model Application to HTTP Protocols

To begin with, we formulate the response time for HTTP over TCP. There are four cases for HTTP over TCP: non-persistent connection of HTTP/1.0, non-persistent connection with parallel connection of HTTP/1.0, persistent connection without pipelining of HTTP/1.1 and persistent connection with pipelining of HTTP/1.1. Next, we describe the response time for HTTP over SCTP which supports the multi-streaming and avoids the head-of-line blocking.

In all the cases, we assume that the web page is composed of one HTML file and *M* referenced objects. When the client receives the HTML file from the server, it parses the HTML file and then sends another requests for the referenced objects based on the protocol specification. As we can see each procedure, all protocols have the same connection setup, request and receipt of the HTML file. Thus, we will not consider the packet loss in this step. In addition, since the size of control and http request packet is very small and only one packet for all protocol, we will not also consider the loss for these packets. Even if we consider the packet loss in the above cases, the delay time due to the loss will be same for all protocol.

Finally, in the original protocols, the server is required to close the connection after sending the file. But, in this paper, we will not consider the connection close time because it does not affect the comparison of protocols.

A. Non-persistent connection of HTTP/1.0 over TCP

HTTP uses one TCP connection over total transactions and does not need the control channel in both server and client. Flow related to the file transmission in non-persistent connection of HTTP/1.0 is represented in Fig. 2.

Typical TCP connection setup uses the 3-way-hand-shake. HTTP can send the request for HTML file from server in the third packet (GET HTML). Server replies by sending the HTML file. Time to complete these steps is the same for all the protocols. We will denote this initial connection setup and HTML file transfer time as *IT* in the subsequent figures. As can be shown in the Fig. 2, $IT = 2RTT + O/Q + ST$. *ST* represents the slow start time incurred in transferring HTML file. Fig. 2 shows that this protocol needs the connection setup whenever it sends the HTTP request. Thus, mean response time = $IT + M \times (2RTT$ + object transfer time). Here, *M* represents the number of reference objects

Object transfer time (*OT*) can be found by using the Algorithm 1 presented in section 2.

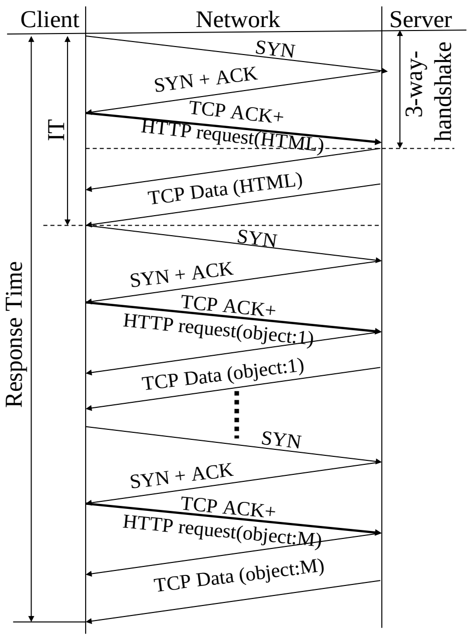

Fig. 2. Non Persistent Connection of HTTP/1.0

B. Non-persitent connection with parallel connection of HTTP/1.0 over TCP

In Fig. 3, we create the *M* processes which use the 3-way-hand-shake simultaneously. *M* processes in the web client can send their requests for *M* objects stored in the web server parallel and receive objects simultaneously. However, The simultaneously created number of processes can be limited by the processing capability of web server. In order to alleviate the creating process load at the client side, we can create multiple threads in a process. However, this method can limit the number of simultaneous threads at a time, which is different by the type of operating system.

Fig. 3 shows that mean response time = $IT + 2RTT$ + object transfer time (OT). The method to find the object transfer time is the same as in the non-persistent connection protocol.

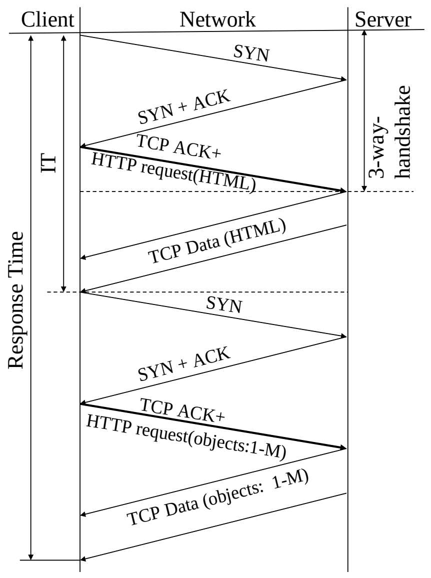

Fig. 3. Non persistent with parallel connection of HTTP/1.0

C. Persitent connection without pipelining of HTTP/1.1 over TCP

HTTP/1.1 can reduce the extra setup time by using the pipelining. But, if we use the pipelining which can affect the performance of the server, the following procedure such as Fig. 4 can be used. In Fig. 4, it is clear that mean response time = $IT + M \times (RTT$ + object transfer time (OT)). We can also find the object transfer time by using Algorithm 1 in section 2.

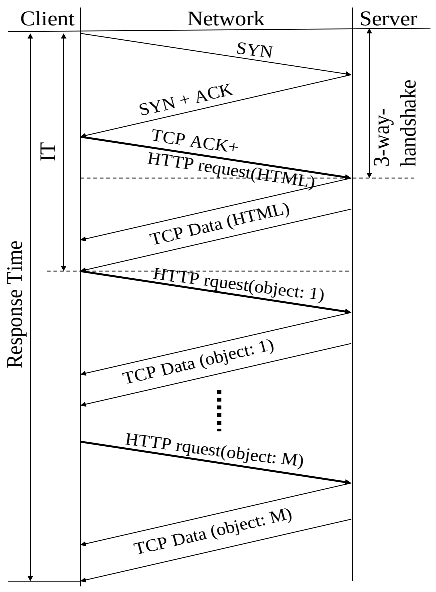

Fig. 4. Persistent connection without pipelining of HTTP/1.1

D. Persitent connection with pipelining of HTTP/1.1 over TCP

Fig. 5 represents the case when the persistent connection with pipelining is used. The maximum number of pipelines to be handled depends on the processing power of server. So, we introduce the variable (L) which represents maximum number of pipelines. If L is greater than or equal to M, then TCP data is transferred at a time. Otherwise, it is transferred at a several times. That is, if $L \geq M$, mean response time = $IT + RTT$ + object transfer time (OT). Otherwise, mean transfer time = $IT + M/L \times (RTT + OT)$. Object transfer time can be found by using Algorithm 1 in section 2.

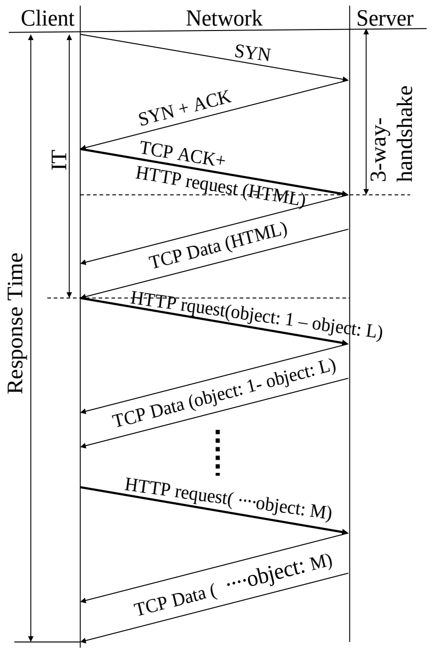
Client
Network
Server
SYN
SYN + ACK
TCP ACK+
HTTP request (HTML)
3-way-handshake
IT
TCP Data (HTML)
HTTP rquest(object: 1 – object: L)
TCP Data (object: 1- object: L)
HTTP request(····object: M)
TCP Data (····object: M)
Response Time

Fig. 5. Persistent connection with pipelining of HTTP/1.1

E. HTTP over SCTP

TCP uses the 3-way-hand-shake connection setup. Instead, the initialization of an association in SCTP is completed after the exchange of four messages. The passive side of the association does not allocate resources for the association until the third of these messages has arrived and been validated. Last two messages of the four-way handshake can already carry user data.

With this piggybacking, SCTP has the same connection-establishment delay as TCP, namely one round trip time. Because SCTP has multi-streaming feature, it can avoid the head-of-line blocking.

Furthermore, it doesn't limit the maximum number of objects which the persistent connection with pipelining of HTTP/1.1 over TCP does. We depict the HTTP over SCTP procedure in Fig. 6. Since HTTP over SCTP does not need the head-of-line blocking time, mean response time = $IT + RTT$ + object transfer time (OT) – hob. Here hob represents the head-of-line blocking time.

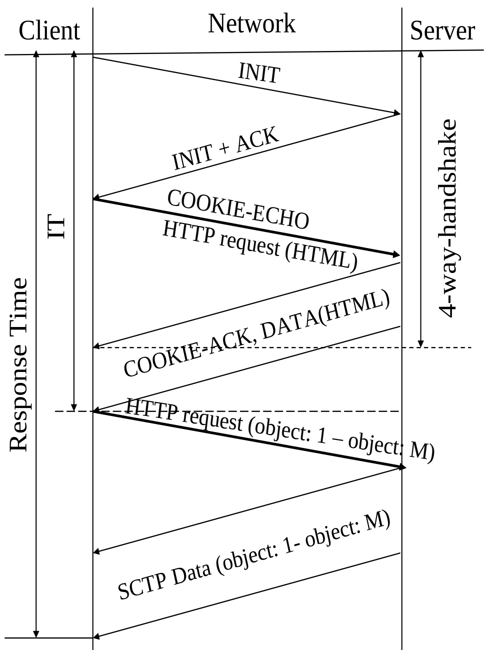

Fig. 6. HTTP over SCTP

IV. Performance Evaluation

We consider the computation complexity of Algorithm 1 in section 2. When the number of packets is N, time complexity of algorithm is O(N).

Step 2 ~ Step 3 of Algorithm 1 can be executed up to the (a + 1) times. When the size of object is O bits and the maximum segment size is b bits, it is clear that N is equal to O/b. Parameter a represents the expected number of packet losses, thereby a is equal to Np. That is, we should compute the expressions (Np + 1) times. All the expressions in each step of algorithm can be computed in O(1). In addition, since we assumed that p is constant, time complexity of algorithm becomes O(N).

Table 1 summarizes mean response times presented in $A \sim E$ of section 3. In Table 1, it is noted that IT is equal to $2RTT + O/Q$. OT and hob are object transfer time and head-of-line blocking time obtained by Algorithm 1 presented in section 2, respectively. Mean response time for HTTP over SCTP is less than that for the enhanced version of HTTP over TCP by head-of-line blocking time.

V. Conclusions

Mean response time is mainly used to measure the delay of end-user in the Internet. To estimate the mean response time is very important in developing and managing the web service and the dimensioning of web server. Since most web services are based on the HTTP over transport layer protocols, we first investigated main elements of mean response time such as the data transfer time, retransmission time, head-of-line blocking time and slow-start time. Since most TCP connections carry short HTTP data, connection setup time and slow-start time are main structural elements of mean response time. Therefore, we derive the analytical model and algorithm for object transfer time including slow-start time and head-of-line blocking time. We then investigated all sort of HTTP transactions over TCP and HTTP over SCTP and estimate approximations of mean response times for every cases. The comparison of mean response times between HTTP over TCP and SCTP shows that mean response time can be reduced by head-of-line blocking time when using HTTP over SCTP. Further works include more exact model and algorithm to cover the entire congestion control mechanism.

Table 1. Comparison of mean response time

application protocol	related figure	transport protocol	mean response time
non-persistent connection of HTTP/1.0	Fig. 2	TCP	$IT + M \times (2RTT + OT)$
non-persistent connection with parallel connection of HTTP/1.0	Fig. 3		$IT + 2RTT + OT$
persistent connection without pipelining of HTTP/1.1	Fig. 4		$IT + M \times (RTT + OT)$
persistent connection with pipelining of HTTP/1.1	Fig. 5		$IT + RTT + OT$, if $L \geq M$ $IT + M/L \times (RTT + OT)$, otherwise
HTTP	Fig. 6	SCTP	$IT + RTT + OT - hob$

References

[1] T. Berners-Lee, R. Fielding and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0", RFC-1945, 1996.

[2] H. F. Nielson and J. Gettys, "Network Performance Effects of HTTP/1.1, CSS1, and PNG", *ACM*, 1997.

[3] R. Stewart, Q. Xie, et al, Stream Control Transmission Protocol, RFC-2960, 2000.

[4] A. L. Caro, J. R. Iyengar, P. D. Amer, S. Ladha, G. Heinz and K. Shah, "SCTP: A Proposed Standard for Robust Internet Data Transport", IEEE Computer, Vol. 36, No. 11, pp. 56-63, 2003.

[5] S. Fu and M. Atiquzzaman, "SCTP: State of the art in Research, Products, and Technical Challenges", *Proc. of IEEE 18th Annual Workshop on Computer Comm.*, October, 2003.

[6] J. Padhye, V. Firoiu, D. F. Towsley and J. F. Kurose, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation", *ACM Transactions on Networking*, Vol. 8, No. 2, pp. 133-145, 2000.

[7] B. A. Mah, P. Sholander, L. Martinez and L. Tolendino,

"IPB: An Internet Protocol Benchmark using Simulated Traffic", *Proc. of 6th International Symposium on Modeling, Analysis and Simluation of Computer and Telecommunication Systems*, pp. 77-84, 1988.

[8] N. Cardwell, S. Savage and T. Anderson, "Modeling TCP Latency", *Proc. of IEEE Infocom*, pp. 1742-1751, 2000.

[9] Z. Jiong, Z. Shu-jing and Qi-gang, "An Adapted Full Model for TCP Latency", *Proc. of IEEE TENCON '02*, pp. 801-804, 2002.

[10] Lin-Huang Chang, Ming-Yi Liao and De-Yu Wang, "Analysis of FTP over SCTP in Congested Network", *Proc. of 2007 International Conference on Advanced Information Technologies (AIT)*, 2007, pp. 82-89.

[11] Chia-Wen Lu and Quincy Wur, "Performance study on SNMP and SIP over SCTP in wireless sensor networks", *Proc. of 14th International conference on advanced communication technology (ICACT)*, pp. 844-847, 2012.

[12] Fei Ge, Liansheng Tan, Jinsheng Sun, and Moshe Zukerman, "Latency of fast TCP for HTTP transactions", *IEEE Communications Letters*, Vol. 15, No. 11, pp. 1259-1261, 2011.

[13] J. Eklund, K. Grinnemo, A. Brunstorm, G. Cheimnidis, and Y. Ismailov, "Impact of Slow Start on SCTP Handover Performance", *Proc. of the 20th international conference on computer communications and networks*, pp. 1-7, 2011.

[14] Y. –J. Lee, M. Atiquzzaman and S. K. sivagurunathan, "Mean Response Time Estimation for HTTP over SCTP in Wireless Environment", *Proc. of IEEE ICC 2006 Conference*, Istanbul, Turkey, 2006.

[15] Y. –J. Lee, "Mean Response Delay Estimation for HTTP over SCTP in Wireless Internet", *Journal of the Korea Contents Association*, Vol. 8, No. 6, pp. 43-53, 2008.

[16] K. W. Ross and J. F. Kurose, *Computer Networking*, Pearson Education, 2012.

Authors' Profiles

Dr. Y. –J. Lee is currently serving as a Professor in the Department of Technology Education, Korea National University of Education, Cheongju, South Korea. His research interests include Internet Technology, Mobile Computing and Performance Evaluation.

How to cite this paper: Y. –J. Lee,"Mean Response Time Approximation for HTTP Transactions over Transport Protocols", IJCNIS, vol.7, no.1, pp.24-30, 2015. DOI: 10.5815/ijcnis.2015.01.04