IEICE TRANSACTIONS

on Communications

VOL. E98-B NO. 9
SEPTEMBER 2015

A PUBLICATION OF THE COMMUNICATIONS SOCIETY

The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3chome, Minato-ku, TOKYO, 105-0011 JAPAN

IEICE TRANS. COMMUN., VOL.E98–B, NO.9 SEPTEMBER 2015

PAPER

RTT Estimation with Sampled Flow Data

Qi SU[†,††a)], **Jian GONG**[†,††b)], *Nonmembers*, *and* **Xiaoyan HU**[†,††c)], *Student Member*

SUMMARY Round-trip time (RTT) is an important performance metric. Traditional RTT estimation methods usually depend on the cooperation of other networks and particular active or passive measurement platforms, whose global deployments are costly and difficult. Thus a new RTT estimation algorithm, ME algorithm, is introduced. It can estimate the RTT of two hosts communicating through border routers by using TCP CUBIC bulk flow data from those routhers without the use of extra facilities, which makes the RTT estimation in large-scale high-speed networks more effective. In addition, a simpler and more accurate algorithm—AE algorithm—is presented and used when the link has large bandwidth and low packet loss rate. The two proposed algorithms suit sampled flow data because only duration and total packet number of a TCP CUBIC bulk flow are inputs to their calculations. Experimental results show that both algorithms work excellently in real situations. Moreover, they have the potential to be adapted to other TCP versions with slight modification as their basic idea is independent of the TCP congestion control mechanism.

key words: *RTT estimation, sampled flow data, network measurement, network management, Markov chain*

1. Introduction

RTT is generally defined as the interval between the timestamp of a sent packet and that of the corresponding acknowledgment. This is an essential metric and an important indicator for network performance monitoring.

The requirements of RTT estimation for various applications are different. Some applications need the measurement with high degree of precision, such as the applications for high performance computing tasks within data center networks and those for electronic trading systems. There are particular solutions that use specialized facilities to meet this requirement, such as [1]–[3]. Diagnosis or troubleshooting oriented applications, on the other hand, need on-demand RTT measurement; thus, inefficient estimation may be acceptable, and the scalability of the estimation methods is not a critical concern. However, most of the other applications, particularly those focusing on performance assessment, are more interested in tracking whether RTT values are within a specified range or have exceeded a threshold for a long time, such as the application of monitoring an ISP's QoS by the RTT values between the hosts inside and outside the ISP and the application of evaluating the user experience of an ICP (Internet Content Provider) based on the RTT values from users' hosts to it. These applications can tolerate approximate estimation results and a certain amount of error. However, efficiency and scalability are the major concerns for their RTT estimation because they usually perform a long-term monitoring for large-scale high-speed networks, and it is insufficient to conduct these applications based on the RTT values from a small number of host pairs in short duration. This paper makes an attempt to explore a method that ensures efficiency and scalability of RTT estimation in large-scale high-speed networks.

RTT estimation has drawn numerous researchers' attention and efforts. Their methods can be broadly divided into two categories—active ones and passive ones. Of the former, taking advantage of the interval between sending an ICMP echo request and receiving corresponding ICMP echo reply is the most prevalent one, and using the interval between sending TCP SYN packet and receiving the first ACK packet is another approach. Despite the high accuracy these methods can achieve, their results can only be collected in end-hosts. Nevertheless, RTT estimation, already an intrinsic part of the TCP, is trivial at end-hosts, while it is extremely challenging yet valuable in the middle of the networks [4], which is the common environment of network management. Besides, these methods can hardly be deployed in large-scale networks that have massive hosts. What's more, due to the increasingly significant network security issues, non-cooperative operators of other networks tend to block the long-lasting measurement traffic, making sustainable monitoring almost impossible.

The passive methods, on the other hand, use real traffic for estimation. Among these methods, SA and SS [5] are the most classic ones. SA employs the interval between the last SYN packet and the first ACK packet from caller to callee to estimate RTT. The basic idea of SS is to regard the interval of the first and the second burst of the TCP session as the RTT value. Though simple and accurate, these two methods are of limited utility because only RTT values of the beginning are obtained, and they can fluctuate considerably over the transmission. Zhang et al. [6] propose a method that generates a series of RTT value candidates, and the one that fits the behavior of current TCP session best is chosen to estimate the real RTT. The authors in [7] calculate the congestion window (cwnd) size, then use it to find the two packets–the receiver's acknowledgment packet and the one that triggers it. The interval between these two

Manuscript received February 9, 2015.
Manuscript revised May 19, 2015.

[†]The authors are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, P. R. China.
[††]The authors are with the Jiangsu Provincial Key Laboratory of Computer Network Technology, Nanjing 211189, P. R. China.
a) E-mail: qsu@njnet.edu.cn
b) E-mail: jgong@njnet.edu.cn
c) E-mail: xyhu@njnet.edu.cn

DOI: 10.1587/transcom.E98.B.1848

packets' timestamps is considered as the estimation result. The ALE algorithm introduced in [4] is an approximate and scalable RTT estimation method. Instead of recording the actual timestamps of the packets, this algorithm maps the timesamps into discrete intervals with fixed or exponential sizes and uses the middle value of intervals to estimate RTT. The above-mentioned methods are based on full packets. Nevertheless, capturing or tracking them in large-scale high-speed networks demands high-end solutions, which are usually costly and require extra measurement facilities and platforms. In addition, the collected data needs a lot of storage to keep and computation power to process, making the routine network management even harder. Therefore, none of those methods is capable of practical, sustainable and comprehensive monitoring of the network performance.

Flow data is another available source data for RTT estimation. What is more, flow data based estimation needs no extra collection facilities and can be deployed easily because most extant routers provide mechanisms to capture flow data, like NetFlow (cf. RFC 3954) and sFlow (cf. RFC 3176), and a corresponding Internet standard, IPFIX (cf. RFC 5101), has been introduced by IETF. Based on flow data, Strohmeier et al. [8] propose a measurement model in which two related opposite-direction TCP flows are carefully selected, and the interval of their first packets' timestamps is used to estimate RTT. However, the scalability and practicality of this method are weakened by the fact that two associated flows need finding and that this method can not be applied in the sampling environment since it uses the first packet's timestamp of each flow. As the amount of traffic grows, the flow data without sampling operation becomes harder to gather and analyze.

To wrap up, given that the extant methods can not fulfill the requirements of the performance monitoring applications for large-scale high-speed networks, we propose a RTT estimation method that can take sampled TCP CUBIC bulk flows as inputs. TCP CUBIC bulk flows are chosen for discussion because they, unlike TCP interactive flows, can be free from users' responsive time and the data processing time (elaborated on in Sect. 3); besides, TCP CUBIC is the default TCP implementation of the Linux (after Linux 2.6.29 [9]), the most popular operation system for the TCP bulk flow senders (e.g. the WEB and FTP servers); as a result, most TCP bulk flows in the Internet employ the TCP CUBIC congestion control mechanism. After studying the behavior of TCP CUBIC bulk flows, we establish ME algorithm. Meanwhile, we use a simpler yet more accurate algorithm, AE algorithm, when the link between two hosts has low packet loss rate and large bandwidth. These two algorithms are capable of taking sampled flows as the source data because only the duration and total packet number of a TCP CUBIC bulk flow are the inputs. Sampled flow data can be easily collected by nearly all extant routers without extra facilities; therefore, the proposed method can effectively monitor and manage the performance of most networks, especially the large-scale high-speed networks, such as backbone networks and ISP's networks.

The contributions of this paper are summarized as follows:

- We, for the first time, present a method to estimate RTT in sampling environment, and it takes sampled flow data, which is widely supported by extant routers as input. In contrast, traditional RTT estimation methods can hardly be applied to the performance monitoring of large-scale high-speed networks due to the costly high-end solutions and authority limits.
- We modify the old models of TCP CUBIC throughput so as to make them adapt to SACK mechanism and introduce ME algorithm to estimate RTT.
- We present AE algorithm to estimate RTT under its application condition AEC, indicating the link has large bandwidth and low packet loss rate.
- We demonstrate that the proposed method has the potential to be applied to AIMD TCP versions with slight modification.

The remaining paper is organized as follows. In Sect. 2, we illustrate what a TCP CUBIC bulk flow will react when packet loss happens under different circumstances. Based on that, Sect. 3 presents a method with two RTT estimation algorithms, and it is evaluated in Sect 4. Finally, a conclusion is given in Sect 5.

2. The Dynamics of TCP CUBIC Flow

When `Sender` using TCP CUBIC sends data to `Receiver`, there are two very important buffers involved in networking stack—the application buffer and the socket buffer. `Sender` copies data from application buffer to socket buffer and injects packets into network with the TCP CUBIC congestion control algorithm. `Receiver` gets the packets from network and stores them in socket buffer. Then, these packets are copied to application buffer for further processing. We assume that the socket buffer size of `Sender` is smaller than that of `Receiver`, since the default value of socket buffer size of `Sender` in Linux 2.6.27 is 16KB, whereas that of `Receiver`'s is 85KB [10]. Therefore, in the later discussion of this paper, the impact of `Receiver` will be ignored.

Figure 1 shows how the cwnd size changes when packet loss happens under two different conditions. Cwnd size describes the maximum number of packets that can be sent without any acknowledgment. Let B be the size of the socket buffer, and the maximum size of cwnd W (in packet for convenience) is given by

$$W = \frac{B}{MSS}, \tag{1}$$

where MSS is the maximum segment size. TCP CUBIC session starts from the slow start mode, and we use the traditional slow start algorithm to formulate this mode, rather than the original Hybrid Slow Start algorithm [11] (the reason will be discussed in detail in Sect. 3.4). The cwnd size of the r^{th} round of the slow start mode can be calculated as follows:

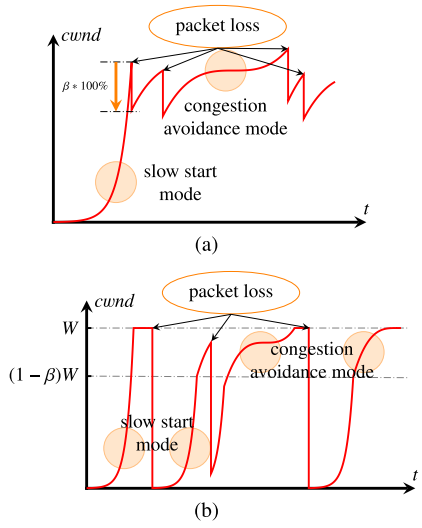


Fig. 1 Typical cwnd size curve for TCP CUBIC with packet loss when cwnd size is (a) far from W and (b) near (or equal to) W.

$$F_{SS}(r) = 2^r. \tag{2}$$

When cwnd size is larger than *ssthresh*, the transmission enters the congestion avoidance mode. In this mode, cwnd size generally grows according to a cubic function:

$$F_{cubic}(w,t) = w + c\left[t - \sqrt[3]{\beta w/c}\right]^3,$$

where w is the cwnd size just before the packet loss happens and t is the time since that loss. c and β in the function are constants that are set to 0.4 and 0.3 respectively in the latest TCP CUBIC. TCP CUBIC also incorporates TCP-friendly behavior into its operation, which means it does not steal bandwidth from standard TCP. To achieve this, TCP CUBIC uses another cwnd growth function in TCP-friendly region [9]:

$$F_{reno}(w,t,\tau) = (1-\beta)w + \frac{3\beta}{2-\beta}\frac{t}{\tau},$$

where τ is the RTT. If $F_{reno}(w,t)$ is larger than $F_{cubic}(w,t)$, the TCP CUBIC operates in TCP-friendly region and uses the former function as its cwnd growth function; otherwise, $F_{cubic}(w,t)$ is employed. Thus, the overall cwnd growth function of the TCP CUBIC evolves as

$$F_{CA}(w,t,\tau) = \max(F_{cubic}(w,t), F_{reno}(w,t,\tau)). \tag{3}$$

As is shown in Fig 1(a), the congestion avoidance mode repeats once a packet loss occurs.

But as Fig. 1(b) shows, when a packet loss happens as the cwnd size is near or equal to the maximum value W, the cwnd size will then drop dramatically, and the transmission will enter the slow start mode rather than the congestion avoidance mode. The SACK mechanism (cf. RFC 2018) is responsible for that. Once a packet loss occurs, at least one packet is not acknowledged; thus, it is impossible to copy more than $W - W'$ data from application buffer to socket buffer, and `Sender` cannot transmit more than $W - W' + 1$ packets until the lost packet is announced. At the same time, as the acknowledgments of other sent packets arrive, the flight size, *in_flight*, calculated by `Sender`, decreases to $W - W' + 1$ and the cwnd size drops accordingly since "$cwnd = \min(cwnd, in\_flight + 1)$". Consequently, when W' is near W (i.e. $W - W' + 1 < (1-\beta)W'$), the resulant cwnd size will be less than $(1-\beta)W'$, and the transmission will step into the slow start mode; this phenomenon, which has also been discussed in [10], will happen if the cwnd size just before the packet loss W' satisfies

$$W - W' + 1 < (1-\beta)W' \Rightarrow W' > (W+1)/(2-\beta). \tag{4}$$

The above discussion shows that the condition of the link between two end-hosts can deeply impact the dynamics of the TCP CUBIC bulk flows transmitting through it. When the condition is not so good (bandwidth is small or packet loss rate is high), the cwnd size will hardly reach the W, and its curve is likely to evolve as Fig. 1(a) shows. On the other hand, if the link condition is reliable (bandwidth is large and packet loss rate is low), the curve of this TCP CUBIC session can be exhibited as Fig. 1(b). Based on that, the next section will introduce the estimation method.

3. RTT Estimation

Figure 2 depicts the basic scenario of the TCP transmission. Three participants are involved, the `Sender`, the `Receiver` and the intermediate network where `Monitor` collects the flow data. The packets sent by `Sender` are grouped burst by burst ($n_1, n_2, \ldots, n_k$ packets in Fig. 2). After a burst of packets arrive at the `Receiver`, it takes t_1 time to process them and then send a burst of ACK packets back to `Sender`; after ACK packets arrive, it takes t_2 time for `Sender` to handle them and start to send another burst of packets. The TCP session transmits all the data by such analogy. In this paper, the procedure where `Sender` sends a burst of packets and receives the corresponding burst of ACK packets is called a *round* (see orange dashed box in Fig. 2). Note that compared with RTT, the intervals of any two data packets within a burst can be ignored; the same is true for any two ACK packets within a burst. Hence, we assume that the sending time of the packets within a burst is almost the same, so is their arriving time.

The essential idea of the RTT estimation in this paper is illustrated as follows. Let $w(r)$ be the burst length of the r^{th} round. If the total packet number during transmission is N, total time consumed is T, and average RTT is τ, we can get

$$\sum_{r=1}^{T/\tau} w(r) = N, \tag{5}$$

where T/τ calculates the number of rounds. Obviously, the RTT value τ can be obtained by (5) once $w(r)$ for each round has been obtained.

As to `Monitor`, it can only sense the interval between

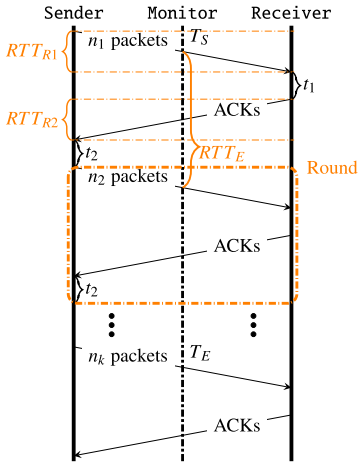


Fig. 2 Basic model of the proposed method.

two adjacent bursts (RTT_E in Fig. 2), and uses it to estimate the real RTT (equals $RTT_{R1} + RTT_{R2}$ in Fig. 2). However, due to the process time of `Sender` and `Receiver` (t_1, t_2 in Fig. 2), RTT_E has certain disparity with the real RTT; hence, we should select the flow data in which the t_1 and t_2 are as short as possible so as to gain higher accuracy. TCP flows can be divided into two classes: TCP interactive flows containing interactive data (like Telnet and Rlogin data), and TCP bulk flows containing bulk flow data (like FTP and HTTP data). This paper selects the bulk flows instead of interactive flows to estimate RTT due to the following reason: for TCP interactive flows, t_1 generally contains the command processing time (Telnet), and t_2 usually includes the time before users finish typing, both of which are usually much larger than RTT values and can not be elided; TCP bulk flows, on the other hand, can avoid the interference of t_1 and t_2, because data processing and users' action are not involved once `Sender` starts to transfer data. Moreover, only TCP CUBIC flows are used as the inputs of the proposed method since the TCP CUBIC is applied to the most popular operation system of TCP bulk flow senders, as mentioned in Sect. 1. To sum up, we choose (sampled) TCP CUBIC bulk flow data collected by intermediate routers (`Monitor` in Fig. 2) as inputs. For a TCP CUBIC bulk flow, suppose that there is always enough data to transmit till the end of transmission, the `Sender` will never stop sending packets until the number of unacknowledged packets reaches current congestion window (cwnd) size; as a result, the (5) will be

$$\sum_{r=1}^{T/\tau} cwnd(r) = N, \tag{6}$$

where $cwnd(r)$ describes the congestion window size of r^{th} round (i.e. $n_1, n_2, \ldots, n_k$ in Fig. 2 are the cwnd sizes of each round).

Based on the modification of the extant TCP CUBIC throughput models in [10], [13], [14], we estimate the cwnd size of every round and present a new RTT estimation method with two algorithms, which are all based on these

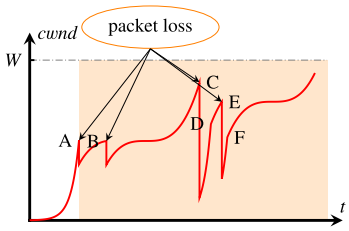


Fig. 3 Basic idea of ME algorithm.

two assumptions:

- The probabilities of all packets experiencing packet loss in the flow are assumed to be identical and mutually independent (the shared probability p, i.e. the packet loss rate, can be calculated by the method presented in [12], which can be applied to sampling environment).
- The RTT values during the transmission are assumed to be constant since our method is aimed to estimation the average RTT.

This section is organized as follows. In the first subsection, ME algorithm is introduced, which generates several RTT candidates and chooses the value that fits the flow best to be the estimated RTT. In addition, AE algorithm, as well as its application condition, is presented in second subsection. Despite its limited application, the AE algorithm is of less complexity and higher accuracy compared with the former algorithm. Then, Sect. 3.3 elaborates on the adaptation of these two algorithms to sampled flow data. Finally, we discuss the reasonableness of replacing Hybrid Slow Start algorithm and the adaption of these two algorithms to AIMD TCP visions respectively in the last two subsections.

3.1 Markov Estimation Algorithm (ME Algorithm)

A typical TCP CUBIC bulk flow is exhibited in Fig. 3. After the first slow start mode, the transmission may enter either the congestion avoidance mode or the slow start mode once a packet loss occurs (the orange area after point A in Fig. 3). As was discussed in Sect. 2, if the cwnd size at that time is larger than $(W + 1)/(2 - \beta)$, the transmission will step into the slow start mode; otherwise, it will enter the congestion avoidance mode.

For a pacrticular round of the transmission, let the cwnd size just before the last packet loss be w and the number of rounds since that loss be r. We can obtain the cwnd size of this round by

$$F_{win}(w, r) = \begin{cases} \min\{F_{CA}(w, r\tau, \tau), W\}, & \text{if } w \le \frac{W+1}{2-\beta} \\ 2^r, & \text{if } w > \frac{W+1}{2-\beta} \ \&\ r \le r' \\ \min\{F_{CA}(w, (r-r')\tau, \tau), W\}, & \text{others} \end{cases}, \tag{7}$$

where W is the maximum of cwnd, τ is the RTT, r' is

$\log_2(1-\beta)w$, and F_{CA} is defined in (3). This function is a piecewise function with three sub-functions:

- The first one describes the situation where the transmission steps into the congestion avoidance mode after a packet loss happens (from point A to point B and point B to point C in Fig. 3).
- The second one takes effect when (4) is fulfilled; the transmission enters the slow start mode until cwnd reaches *ssthread*, see point C to point D and point E to point F in Fig. 3.
- The last one comes into use when (4) is fulfilled and cwnd exceeds *ssthread*, which is illustrated from point D to point E and after point F in Fig. 3.

For a given packet loss rate p, if the cwnd size of a round is w, the probability of no packet losing in this round is $(1-p)^w$, and the one of dropping at least one packet is $1-(1-p)^w$. Considering a stochastic process $\{W_k\}$, W_k equals w if and only if the cwnd size is w just before the k^{th} loss event. If $W_i = w_i$, for $i = 1, 2, \ldots, k-1$, the probability of the k^{th} loss happening after $r (r = 0, 1, \ldots)$ rounds is

$$\begin{aligned}
&\Pr\{W_k = F_{win}(w_{k-1}, r) | W_{k-1} = w_{k-1}, \ldots, W_1 = w_1\} \\
&= (1-(1-p)^{F_{win}(w_{k-1},r)}) \prod_{i=0}^{r-1} (1-p)^{F_{win}(w_{k-1},i)} \\
&= \Pr\{W_k = F_{win}(w_{k-1}, r) | W_{k-1} = w_{k-1}\}, \qquad (8)
\end{aligned}$$

denoted as $p(w_k, r)$. From (8), we can find the process $\{W_k\}$ is a Markov chain with states $\{1, 2, \ldots, W\}$ [15, chap. 4]. The transition matrix of this Markov chain $\mathbb{P} = \{P_{ij}\}$ can be calculated by

$$P_{ij} = \begin{cases} \sum_{k=0}^{r''} p(i,k) F_{zero}(F_{win}(i,k) - j), & \text{if } j < W \\ 1 - \sum_{k=1}^{W-1} P_{ik}, & \text{if } j = W \end{cases}, \qquad (9)$$

where $r'' = \max\{r \mid F_{win}(i, r) < W\}$ and F_{zero} is defined as

$$F_{zero}(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{others} \end{cases}.$$

State 1 of $\{W_k\}$ is reachable from any state within finite steps since $\max\{W_k\} = W < \infty$ and the minimum probability of packet loss is p. It implies that the $\{W_k\}$ is an irreducible positive recurrent Markov chain with a unique stationary distribution $\pi = (\pi_1, \pi_2, \ldots, \pi_W)$, in which π_i is the stationary probability of the state i [14]. From (7)~(9), we can derive the π by solving

$$\begin{cases} \pi\mathbb{P} = \pi \\ \sum_i \pi_i = 1 \end{cases},$$

And the expectation of cwnd size in the orange area of Fig. 3, E_w, is obtained by

$$E_w = \frac{\sum_{i=1}^{W} \sum_{j=1}^{W} \pi_i P_{ij} s_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{W} \pi_i P_{ij} r_{ij}}, \qquad (10)$$

where r_{ij} is the number of rounds it takes for cwnd size to grow from i to j:

$$r_{ij} = \left(\sqrt[3]{(j-i)/c} + \sqrt[3]{\beta i/c} \right) / \tau,$$

and s_{ij} refers to the number of packets sent when cwnd size grows from i to j:

$$s_{ij} = \sum_{r=0}^{r_{ij}} F_{win}(i, r).$$

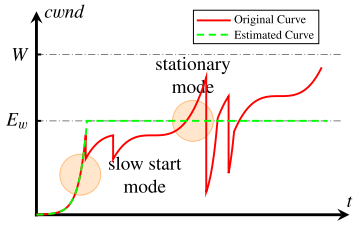


Fig. 4 Illustration of ME algorithm.

As Fig. 4 displays, the estimated packet number of a TCP CUBIC bulk flow N_E can be calculated as follows:

$$N_E = N_{SS} + N_S, \qquad (11)$$

where N_{SS} is the number of packets transmitted during the slow start mode, and N_S is the one during the stationary mode. From (2), N_{SS} can be derived as

$$N_{SS} = \sum_{r=0}^{\log_2 E_w} F_{SS}(r) = \sum_{r=0}^{\log_2 E_w} 2^r = 2E_w - 1. \qquad (12)$$

The time consumed in the slow start mode T_{SS} is

$$T_{SS} = \tau \log_2 E_w. \qquad (13)$$

Therefore, N_S is

$$N_S = (T - T_{SS}) E_w / \tau. \qquad (14)$$

From (11)~(14), we can get the estimation of total packets N_E by

$$N_E = 2E_w + TE_w/\tau - E_w \log_2 E_w - 1, \qquad (15)$$

where T is the total time, E_w is the expectation of cwnd size in (10), and τ is the target RTT.

Drawing the idea from [6], we generate a series of RTT candidates, calculate the corresponding N_E by (15) for each of them and finally choose the RTT candidate whose N_E matches the real packet number N best to be the estimated RTT value.

3.2 Area Estimation Algorithm (AE Algorithm)

The ME algorithm introduced in the previous subsection can be applied to all situations, whereas its calculation is complex. Here we present Area Estimation (AE) algorithm, a simpler and more accurate (demonstrated in Sect. 4.1) estimation algorithm working perfectly under its application

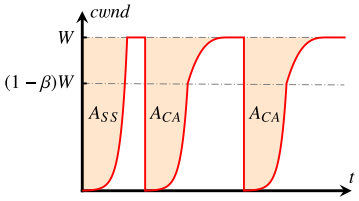


Fig. 5 Illustration of AE algorithm.

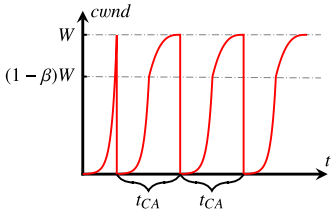


Fig. 6 Maximum packet loss for AE algorithm.

condition — AEC.

For a link with low packet loss rate and large bandwidth, the packet loss rarely happens throughout the TCP sessions, and the cwnd size will quickly grow to W. Consequently, an assumption is made that no packet loss happens until the cwnd size grows to W; in other words, there is no packet loss when the cwnd size grows during slow start mode as well as congestion avoidance mode until cwnd reaches W, see Fig. 5. As a result, an area A_{CA} will be generated once a packet loss event happens after cwnd size reaches W. Hence, we have

$$WT/\tau = N + A_{SS} + NpA_{CA}, \tag{16}$$

where T is the duration of the flow, τ the unknown RTT, N the packet number of the flow, and p the packet loss rate. From (2), A_{SS} can be derived by

$$A_{SS} = \sum_{r=0}^{\log_2 W} (W - F_{SS}(r)) = W\log_2 W - 2W + 1 = L(1), \tag{17}$$

where $L(x)$ is defined as

$$L(x) = W\log_2 xW - 2xW + 1, \tag{18}$$

and the A_{CA} can be given by

$$\begin{aligned} A_{CA} &= \sum_{r=0}^{\log_2(1-\beta)W} (W - F_{SS}(r)) + \int_0^{t'} W - F_{CA}(W,t,\tau)\mathrm{d}\frac{t}{\tau} \\ &= L(1-\beta) + \min\{\frac{\beta}{4\tau}\sqrt[3]{\frac{\beta W^4}{c}}, \frac{(2-\beta)\beta W^2}{6}\}, \end{aligned} \tag{19}$$

where t' makes $F_{CA}((1-\beta)W, t', \tau)$ equal to W. Note that we use $\max\{F_{cubic}(w,t), F_{reno}(w,t,\tau)\}$ to roughly replace $F_{CA}(w,t,\tau)$ for the sake of simplicity. In practice, $\beta/(4\tau)\sqrt[3]{\beta W^4/c}$ is usually smaller than $(2-\beta)\beta W^2/6$; at this time, we can obtain the estimation of RTT τ by

$$\tau = \frac{WT - (\beta Np/4)\sqrt[3]{\beta W^4/c}}{N + L(1) + NpL(1-\beta)}, \tag{20}$$

where $L(x)$ is defined in (18), W is the maximum cwnd size, T the duration of the flow, N the packet number, p the loss rate, and c, β are two constants of TCP CUBIC that are usually 0.4 and 0.3 respectively. Besides, if $\beta/(4\tau)\sqrt[3]{\beta W^4/c}$ is larger than $(2-\beta)\beta W^2/6$, the estimated RTT is

$$\tau = \frac{WT}{N + L(1) + Np(L(1-\beta) + (2-\beta)\beta W^2/6)}. \tag{21}$$

From (20) and (21), we can find that the AE algorithm is much simpler than ME algorithm. It will work excellently when the link is reliable but can cause huge error when the link is of small bandwidth or high packet loss rate. Therefore, we present AEC (Area Estimation Condition), the prerequisite for the application of AE algorithm. Its idea is quite straightforward. If AE algorithm is suitable for this flow, the number of lost packets is certainly smaller than that shown in Fig. 6, which displays the maximum number of dropped packets in the TCP session modeled by AE algorithm. Let the RTT calculated for a flow by AE algorithm in Fig. 6 be τ_e, the t_{CA} can be obtained by

$$t_{CA} = \tau_e \log_2(1-\beta)W + \min\left\{\sqrt[3]{\beta W/c}, (2-\beta)W\tau_e/3\right\}, \tag{22}$$

and the number of lost packet is

$$N'_{loss} = (T - \tau_e \log_2 W)/t_{CA} + 1, \tag{23}$$

where T is the duration of flow. To summarize, AEC is $Np < N'_{loss}$, where N is the total packet number, p is the packet loss rate and the τ_e in the calculation of N'_{loss} is derived by AE algorithm.

According to the evaluation that will be exhibited in Sect. 4.1, we recommend that AE algorithm should be applied under AEC, while ME algorithm should be used otherwise.

3.3 Adaptation to Sampled Flow Data

The proposed method only needs the total packet number N and the total time T, which can be worked out by the total packet number N_s and the total time T_s in a sampled TCP CUBIC flow. As is discussed in [16], when sampling operation is conducted in large-scale high-speed networks, it is reasonable to assume that the sampling decision for every packet in this flow is mutually independent, and the probability of each one being selected is a constant r_s.

The probability of a flow with x packets before sampling and N_s packets after is (let X describe the original packet number, and Y the sampled packet number)

$$\begin{aligned} P(x, N_s) &= \Pr\{X = x | Y = N_s\} \\ &= \frac{\Pr\{Y = N_s | X = x\}\Pr\{X = x\}}{\sum_{x'=N_s}^{\infty} \Pr\{Y = N_s | X = x'\}\Pr\{X = x'\}}, \end{aligned} \tag{24}$$

where

$$\Pr\{Y = N_s | X = x\} = \binom{x}{N_s} r_s^{N_s}(1 - r_s)^{x-N_s}, \tag{25}$$

and the estimation of the original flow length distribution $\Pr(X = x)$ is explained in detail in [16]. Simply, since the flow length follows the heavy-tailed distribution, we can use Pareto distribution with 1 as the parameter to roughly estimate the distribution:

$$\Pr\{X = x\} = \begin{cases} 0, & \text{if } x < N_s \\ \dfrac{\beta N_s^{\beta}}{x^{\beta+1}} = \dfrac{N_s}{x^2}, & \text{if } x \geq N_s \end{cases}. \tag{26}$$

Then, we have the expectation of the original flow length

$$N = \sum_{x=N_s}^{\infty} xP(x, N_s). \tag{27}$$

Finally, we simply use the following formula to estimate T in the two estimation algorithms:

$$T = T_s(N_s + 1)/N_s. \tag{28}$$

3.4 Reasonableness of Applying the Traditional Slow Start Algorithm

The Hybrid Slow Start algorithm [11] used in TCP CUBIC is an improvement of the traditional slow start algorithm, because the former, through estimating the bandwidth, eliminates the packet loss caused by overshooting. However, in this paper, the traditional slow start function is applied instead, which can also be seen in [10]. The reasons are listed as follows:

- Firstly, for ME algorithm, as the transmission lasts longer, the slow start mode becomes more trivial than the stationary mode, and during the stationary mode the determination of either slow start algorithm is not significant because the algorithm only takes effect when the cwnd size is larger than $(W + 1)/(2 - \beta)$, whose probability is very low.
- Secondly, there is little difference between applying the Hybrid Slow Start algorithm and the traditional one to AE algorithm because overshooting rarely occurs on the AEC, on which the bandwidth is large.
- Thirdly, it is simpler to use this function in those two algorithms.

3.5 Modification for AIMD TCP Versions

Our method could be adapted to AIMD TCP versions after slight modification. The major difference between AIMD TCP versions and TCP CUBIC lies in their congestion avoidance mode. When there is no packet loss in the previous round, the cwnd size of the next round will increase by a fixed value a; on the other hand, if there is at least one packet loss during the round, the cwnd size will decrease by a multiplicative factor b $(0 < b < 1)$. Thus, the function F_{CA} in (3) will be

$$F_{CA}(w, t) = bw + at/\tau, \tag{29}$$

where τ is the average RTT value. The (7) will accordingly be (without SACK mechanism)

$$F_{win}(w, r) = \min(bw + ar, W), \tag{30}$$

and the (19) will be

$$\begin{aligned} A_{CA} &= \sum_{i=1}^{\log_2 bW} (W - F_{SS}(i)) + \int_0^{\frac{W-bW}{a}\tau} W - F_{CA}(w, t)\mathrm{d}\frac{t}{\tau} \\ &= L(b) + (1 - b)^2 W^2/(2a), \end{aligned} \tag{31}$$

and the (22) can be modified to

$$t_{CA} = \tau_e \log_2 bW + \tau_e(1 - b)W/a. \tag{32}$$

Now we can conduct those two algorithms just by the procedures introduced. In fact, it can be found that under this circumstance, the E_w calculated in (10) will be irrelevant to the average RTT τ, and based on (11) we can easily obtain τ (just change the N_E of the equation to the total packet number N).

4. Evaluation

Sections 4.1 and 4.2 describe the experiments based on generated flows and real trace respectively, and illustrate their results.

During the evaluation, the results of the two algorithms are compared with those of `tcptrace` [17], a well-established tool that can estimate RTT by taking bidirectional trace collected in the middle of the link as input [4]. This tool can calculate RTT values very precisely because the RTT can only be found when an ACK packet P_A is received from the other end-host for a previous transmitted packet P_B, and the acknowledgment value of the P_A is 1 greater than the last sequence number of the P_B. Then, the RTT values during the whole session of this flow are averaged, and the result serves as the baseline result for evaluation.

4.1 Testbed Evaluation

Based on the flow data generated by the testbed deployed in a real network, this section presents experimental results to confirm the validity of ME and AE algorithm. The structure of the testbed is depicted in Fig. 7. All the data communicating with the `Server` is forwarded by the `Router` and collected by the `Monitor`.

The `Server` is connected to three clients: `ClientA`, `ClientB` and `ClientC`. `ClientA`, `ClientB` and `Server` are situated separately in three PoPs of one AS in China—CERNET [18]. The links between them have large bandwidth and rare packet loss; hence, both AE algorithm and

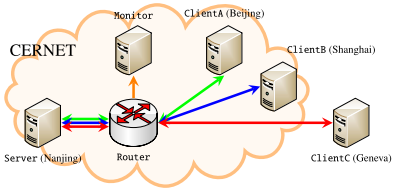
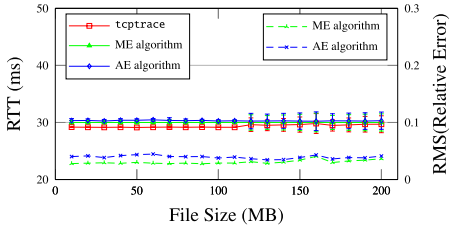
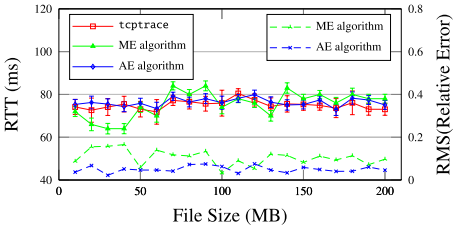


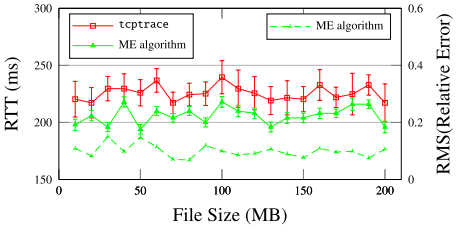
Fig. 7 Illustration of the constructed testbed.



(a) Results of the Flows from `Server` to `ClientA`



(b) Results of the Flows from `Server` to `ClientB`



(c) Results of the Flows from `Server` to `ClientC`

Fig. 8 Testbed results for different file sizes.

ME algorithm can be used under these circumstances. Nevertheless, `ClientC`, a host located in Geneva, Switzerland, communicates with the `Server` through a link with small bandwidth and high packet loss rate, so ME algorithm can only be applied. The RTT candidates of ME algorithm during the evaluation are generated every 10 ms.

During the experiment, the clients (`ClientA`, `ClientB` and `ClientC`) use FTP to get files of various sizes (10 MB, 20 MB, ..., 200 MB) from `Server`. Files of each size are transferred several times and the results are averaged so as to eliminate random error. Figure 8 exhibits the estimation results based on the three groups of TCP CUBIC bulk flows from `Server` to the three clients, with the error bars representing the 95% confidence interval of the average, and we also present the RMS(relative error) for each file size. In addition, Fig. 9 depicts the CDF (Cumulative Distribution Function) of relative error to show the distribution of estimation results over all the flows with different file sizes.

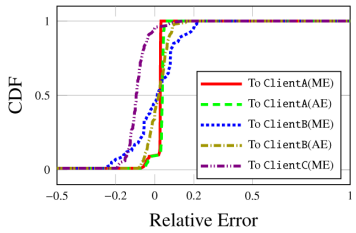


Fig. 9 CDF results of testbed for all file sizes.

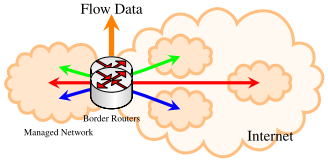


Fig. 10 Deployment demonstration of the proposed method.

The results in the Fig. 9 are ideal because the relative error of nearly all estimations is within ±0.2. What's more, compared with ME, AE algorithm obtains much more accurate results from the flows from `Server` to `ClientB`. We can find that once the AEC presented in Sect. 3.2 is fulfilled, AE algorithm is more suitable for estimation on account of its simplicity and accuracy. As a result, in our method we use AE algorithm instead of ME algorithm under the AEC.

Figures 8(b) and 8(c) show that as the file size (i.e. flow length) increases, the RMS(relative error) of ME algorithm becomes more stable. This phenomenon accords with expectation, because long flows certainly reduce the randomness and volatility of estimation results, and result in the relative error distributed near a small value. As is presented in Fig. 8(c), there is a gap between the curves of two averaged RTT; in other words, smaller RTT values are achieved from nearly all flows from `Server` to `ClientC`, as a result of the large estimated packet number closer to the original packet number in this circumstance, originating from the fact that N_E in (15) does not change linearly with the increase of candidate RTT τ, according to our investigation into the intermediate results. We think that the wise choosing of RTT candidates will overcome this problem and improve the precision of ME algorithm. We will dig into it in the future work.

4.2 Real Trace Evaluation

The typical application scenario of our method is demonstrated in Fig. 10. All traffic of the managed network communicating with other parts of the Internet is forwarded by border routers, where flow data is collected and selected for RTT estimation.

For evaluation, with the help of some additional facilities, we capture full packet bidirectional trace from the border routers located on the border of a regional academic network of the CERNET. With one-hour duration, the capture facility collects roughly 128 GB files, namely 2.70 G packets. To simulate the sampling operation for flow data, we

use a sampling technique which is widely applied in extant routes — systematic count-based sampling (cf. RFC 5476). It is defined as selecting one packet out of every x packets (denoted as "1 out-of x", where x is the sampling period), i.e. $m^{th}, (m+x)^{th}, (m+2x)^{th}, \ldots$ are selected, where m is a randomly chosen value between 1 and x. So, the r_s used in Sect. 3.3 is $1/x$.

Our experiment has four steps:

i) The packets from the trace are grouped into flows after sampling (in practice, this step is accomplished by routers). In this evaluation, four different sampling periods—1 (full-packet), 4, 16 and 64 are used with $m = 1$. Note that all the packets in every flow sampled by period 64 will appear in other three flows sampled by period 16, 4 and 1 respectively. The reason is that for these four sampling periods with the same m ($m = 1$), the systematic count-based sampling ensures that once chosen by a larger sampling period, the packet will certainly be chosen by a smaller one[†]. The related four flows form a 4-tuple, denoted as a F^4.

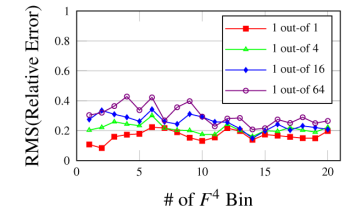
ii) Then, we find all the Linux hosts that use TCP CUBIC, and filter the F^4 s that are not sent from those hosts with certain ports (like 20 for FTP-DATA, 80 for HTTP and 443 for HTTPS). The flows of the left F^4s are TCP CUBIC bulk flows and can be used to estimate RTT by our method.

iii) From 100 to 2100, we construct 20 bins (intervals) with the same length of 100, i.e. the interval of the #1 bin is [100,200), the one of #2 bin is [200,300), ..., and lastly, the one of #20 is [2000,2100). We call them F^4 bins. Then, for each bin, we find out all the F^4s whose full-packet flow length is within the interval of this bin and randomly select 100 from them into this bin.

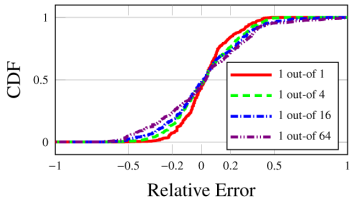
iv) We get the baseline RTT of each F^4 by the `tcptrace` based on full-packet bidirectional trace and compare it with the estimated RTT values of the four flows in this F^4 respectively. According to our method, we use AE algorithm if its requirement—AEC—is fulfilled; otherwise, we use ME algorithm with RTT candidates generated for every 10 ms.

Figure 11(a) shows the evaluation results. The horizontal axis represents the serial number of F^4 bins. Obviously, the larger the bin number is, the longer the flows in that bin are. The vertical axis shows the RMS(relative error) of the four flow groups with different sampling periods from the 100 F^4s in each F^4 bin. We can find that the estimation accuracy decreases as the sampling period rises (can also be seen in Fig. 11(b)), while generally increases with flow length growing. To some extent, long flows can eliminate the error caused by sampling. Besides, we believe that taking more data into estimation is another way to reduce the imprecision. As is exhibited in Fig. 12, more flows used for estimation can relieve the fluctuation of the curve and increase accuracy at the same time.

[†]For example, with $m = 1$, the 65^{th} packet will be selected by the sampling operation with period 64, and it will also be selected when the sampling period is 1, 4, and 16.

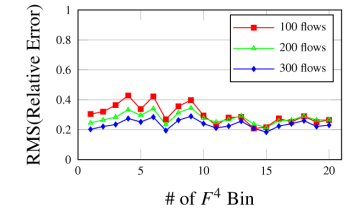


(a) RMS(Relative Error) for Different Sampling Periods

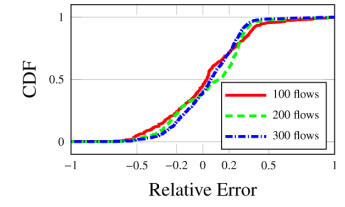


(b) CDF for Different Sampling Periods

Fig. 11 Results of real trace grouped by different sample periods.



(a) RMS(Relative Error) for Different Number of Flows



(b) CDF for Different Number of flows

Fig. 12 Results of real trace grouped by different flow number (1 out-of 64).

5. Conclusion

This paper analyzes the transmission features of the (sampled) TCP CUBIC bulk flow to propose a new RTT estimation method for long-term performance monitoring. The new method taking sampled flow data as input has many advantages. First and foremost, it does not need extra network data collection facilities since sampled flow data is supported by most extant routers; besides, it does not require the cooperation of other networks; moreover, sampling operation in nature ensures that the flow data is easy to collect, store and process. The method comes with two algorithms—ME algorithm and AE algorithm. The former is capable of handling all kinds of TCP CUBIC bulk flows;

the latter is simpler and more accurate but can only be applied under AEC, indicating the bandwidth of the link is large and packet loss rate is low. Moreover, the fact that this method has the potential to be applied to AIMD TCP versions with slight modification proves its significant practicality.

The results of our experiments presented in Sect. 4 demonstrate that RTT estimation can be accomplished with our method. As the flow length grows, the error of the two algorithms trends to be distributed near a small value; in addition, the random error caused by sampling can be eliminated if the number of flows is large enough. These two conditions can be easily fulfilled in large-scale high-speed networks. Consequently, RTT can be measured comprehensively and continuously in these networks by the proposed method.

Our future work will focus on the applications of the calculated RTT values, which will benefit further management tasks.

Acknowledgment

This work is sponsored partly by the National Basic Research Program of China (973) under Grant No. 2009CB320505 and the National Natural Science Foundation of China under Grant No. 60973123.

References

[1] R.R. Kompella, K. Levchenko, A.C. Snoeren, and G. Varghese, "Every microsecond counts: Tracking fine-grain latencies with a lossy difference aggregator," Proc. ACM SIGCOMM 2009 Conference on Data Communication, vol.39, no.4, pp.255–266, 2009.
[2] M. Lee, N. Duffield, and R.R. Kompella, "Not all microseconds are equal: Fine-grained per-flow measurements with reference latency interpolation," Proc. ACM SIGCOMM 2010 Conference on SIGCOMM — SIGCOMM'10, pp.27–38, 2010.
[3] Corvil, "Inter-party latency," 2014. http://corvil.com/solutions/electronic-trading/inter-party-latency
[4] S. Gangam, J. Chandrashekar, Í. Cunha, and J. Kurose, "Estimating TCP latency approximately with passive measurements," Proc. Passive and Active Measurement, Lecture Notes in Computer Science, vol.7799, pp.83–93, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
[5] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," SIGCOMM Comput. Commun. Rev., vol.32, no.3, pp.75–88, 2002.
[6] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates," SIGCOMM Comput. Commun. Rev., vol.32, no.4, pp.309–322, 2002.
[7] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP connection characteristics through passive measurements," Proc. IEEE INFOCOM 2004, pp.1582–1592, 2004.
[8] F. Strohmeier, P. Dorfinger, and B. Trammell, "Network performance evaluation based on flow data," Proc. 2011 7th International Wireless Communications and Mobile Computing Conference, pp.1585–1589, 2011.
[9] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," ACM SIGOPS Operating Systems Review, vol.42, no.5, pp.64–74, 2008.
[10] N. Tomita and S. Valaee, "Data uploading time estimation for CUBIC TCP in long distance networks," Computer Networks, vol.56, no.11, pp.2677–2689, 2012.
[11] S. Ha and I. Rhee, "Hybrid slow start for high-bandwidth and long-distance networks," Proc. PFLDnet, pp.1–6, Manchester, UK, 2008.
[12] Y. Yamasaki, H. Shimonishi, and T. Murase, "Statistical estimation of TCP packet loss rate from sampled ACK packets," Proc. GLOBECOM'05. IEEE Global Telecommunications Conference, 2005, pp.276–280, 2005.
[13] W. Bao, V.W.S. Wong, and V.C.M. Leung, "A model for steady state throughput of TCP CUBIC," Proc. 2010 IEEE Global Telecommunications Conference GLOBECOM 2010, pp.1–6, 2010.
[14] S. Poojary and V. Sharma, "Analytical model for congestion control and throughput with TCP CUBIC connections," Proc. 2011 IEEE Global Telecommunications Conference — GLOBECOM 2011, pp.1–6, 2011.
[15] S.M. Ross, Introduction to Probability Models, 10th ed., Academic Press, 2010.
[16] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," Proc. 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications — SIGCOMM'03, pp.325–336, 2003.
[17] tcptrace, "tcptrace," 2014. http://www.tcptrace.org
[18] CERNET, "China education and research network," 2014. http://www.edu.cn/english_1369/index.shtml

Qi Su is a Ph.D. candidate in School of Computer Science and Engineering, Southeast University, Nanjing, P. R. China. His research interests are network measurement and network management. He received B.S. degree in computer science and technology from the Southeast University, Nanjing, P. R. China.

Jian Gong is a professor in School of Computer Science and Engineering, Southeast University. His research interests are network architecture, network intrusion detection, and network management. He received B.S. degree in computer software from Nanjing University, Nanjing, P. R. China, and Ph.D. degree in computer science and technology from Southeast University, Nanjing, P. R. China.

Xiaoyan Hu got her B.S. degree in software engineering from Nanjing University of Science and Technology in 2007 and her M.S. degree in computer architecture from Southeast University in 2009. She is now a Ph.D. candidate in Southeast University focusing on the design of NDN in-network caching.