On the Correspondency between TCP Acknowledgment Packet and Data Packet

Guohan Lu
Dept. of Electronic Engineering
Tsinghua University, 100084
P.R. China
lvguohan@tsinghua.org.cn

Xing Li
Dept. of Electronic Engineering
Tsinghua University, 100084*
P.R. China
xing@cernet.edu.cn

ABSTRACT
At the TCP sender side, the arrival of an ack packet always triggers the sender to send data packets, which establishes a correspondency between the arrived ack packet and the sent data packets. In a TCP connection, the correspondency between every ack packet and its corresponding data packets forms a sequence. This sequence characterizes the sender's behavior. In this paper, we propose a method to estimate this correspondency sequence from the dump trace measured at the receiver side. Because many possible correspondency sequences can be constructed based on the trace, the problem here is an estimation problem, which is to select a most possible one from those candidate sequences. The method proposed first eliminates some candidates that violate basic TCP congestion behavior. Then, it chooses the most possible one among the remaining sequences using the statistical characteristics of delays between the acks and their corresponding data packets under maximum-likelihood criterion. The method can work in the condition when the TCP connection experiences various network delay and loss, and it applies to TCP senders of different versions. Simulations and Internet experiments have been performed to validate the method.

Categories and Subject Descriptors
C.2.6 [**Computer Communication Networks**]: Internetworking

General Terms
Measurement

Keywords
TCP, Maximum-Likelihood Estimation, non-deterministic Finite State Machine

1. INTRODUCTION
As a window-based congestion control protocol, one basic TCP sender behavior is that its data sending events are nearly all triggered by the arrival of acknowledgement packets, with the only exception of those timeout retransmissions. This establishes a corresponding relation between the ack packet and the data packets it subsequently liberate. For example, in Figure 1, ack packet *A*5 corresponds to data packets *D*7, *D*8, ack *A*7 corresponds to *D*10. In a TCP connection, every ack packet corresponds to a certain number of data packets, which forms a sequence of correspondency between ack and data packets. This correspondency sequence fully characterizes the sender behavior of a TCP, which reveals lots of valuable information:

- The sender's internal congestion states, such as *cwnd*, *ssthresh*, etc on.
- The TCP implementation of the sender. Different TCP implementations have different reactions to arrived ack packets.

Now, we passively measure a TCP connection at its receiver side, and capture all passing ack and data packets. Then, can we deduce the correspondency sequence between the ack and data packet from the dump trace?

Obviously, the data packet liberated by an ack packet must be recorded after the ack in the trace. We call this the time causality condition. However, there are many data packets behind a certain ack packet. Therefore, based on the dump trace, there are many possible correspondency sequences if only time causality condition is satisfied. The correct correspondency sequence is one of them. Now, the problem here is to find this correct one among the candidate sequences.

In this paper, we propose a method to solve this problem. In the method, we first use rules to eliminate some correspondency sequences that violate basic TCP congestion behaviors. Suppose Θ is the set of the remaining sequences. Then, we select one sequence in Θ as the result based on the statistical characteristics of delays between the acks and their corresponding data packets under maximum-likelihood(ML) criterion.

There are two keys in this method: The first key is to use rules to eliminate invalid correspondency sequences. There are many sequences if only time causality condition is satisfied, but many of them do not behave like a TCP, and should be eliminated. In order to eliminate these sequences, we create a model to model the

* This work is supported by cisco university research project.

internal congestion state transition behaviors of a TCP sender. The model is used to check whether the implicit sender's internal state transition path of a correspondency sequence satisfies the basic TCP congestion behaviors. All sequences that are not accepted by this model will be eliminated at this stage. So, the set of those remaining sequences Θ is comprised of only those correspondency sequences which behave like a TCP, and the correct one will be in Θ.

The second key is the maximum-likelihood estimation. First, let us consider the *ta* delay which denotes the interval from the time when the ack packet passes the measure point to the time when its corresponding data packets pass the measure point(see Figure 1). ta_7 and ta_{10} represent the *ta* of data packet *D*7 and *D*10 respectively. In Θ, every correspondency sequence determines a *ta* sequence. Each *ta* can be modeled as a random variable. The *ta* sequence is a stochastic process. Usually, *ta* varies itself within a certain range and correlation exists in the *ta* sequence. So, if we have a stochastic model of *ta* sequence, we can calculate the likelihood value for every *ta* sequence in Θ, and select the one with the maximum likelihood value. The correspondency sequence of this *ta* sequence is the result of our method.

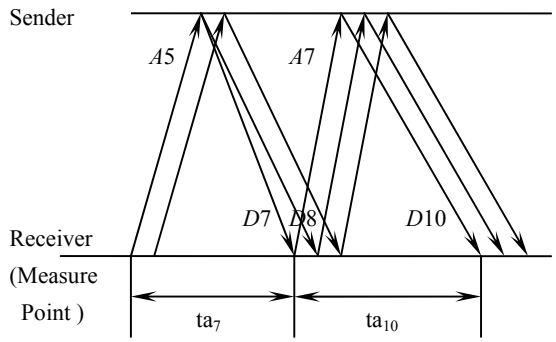

Figure 1 Correspondency between TCP ack and data packet

The method is a statistical estimation, so it cannot guarantee the selected one is the actual correspondency sequence. Based on the method, we implemented a tool *tcpdep* that performs the estimation using the dump trace measured at the receiver side. Through simulations and Internet experiments, we find *tcpdep*'s estimation is of high accuracy under a variety of network environments.

Paper is organized as follows: §2 describes the estimation model. §3 describes the some methods used by *tcpdep*. §4 is the model validation with simulations and experiments. Finally, we draw the conclusion.

1.1 Related Work

Vern Paxson in [13] developed *tcpanaly*. If the sender's behavior of a TCP implementation is going to be analyzed, the packet trace must be captured from a vantage point at or near the TCP sender. *tcpdep*, however, analyzes the pairing relations between acks and data packets without requiring the trace be gathered at or near the sender side.

Zhang et al in [19] use T-RAT to analyze the flow rates of TCP connections. In T-RAT, heuristics are proposed to estimate RTT and to track TCP dynamics. The idea is similar to *tcpdep*. However, methods used in the two tools are different.

Padhye et al in [12] use TBIT to actively examine the TCP implementation of a remote web server by fabricating TCP packets and send them to the remote web server. *tcpdep* is a passive analysis tool, and it tries to pair up acks with the corresponding data packets they subsequently liberate.

Paul Barford et al in [4] use *tcpeval* tool to analyze TCP critical path. The analysis of *tcpeval* requires traces be gathered at both the sender's and receiver's sides, and the goal is to analyze different delays in a TCP connection.

2. ESTIMATION MODEL

Problem: Record a TCP connection at the receiver side[1] (See Figure 2), capture all ack packets sent from B and all data packets arriving at B, and record the time when it is captured. Then, based on this dump trace, we try to estimate the correspondency sequence *R* between the acks and data packets of this TCP connection:

$$R = \{ r_k \mid k = 1,2,\ldots n \},$$

where r_k represents the pair of A_k and its corresponding data packets. $r_k = (A_k; \text{NULL}, [D_j, D_{j+1}, .. D_{j+J}])$. NULL represents that A_k does not correspond to any data packet, $[D_j, D_{j+1}, .. D_{j+J}]$ represent the corresponding data packets of A_k.

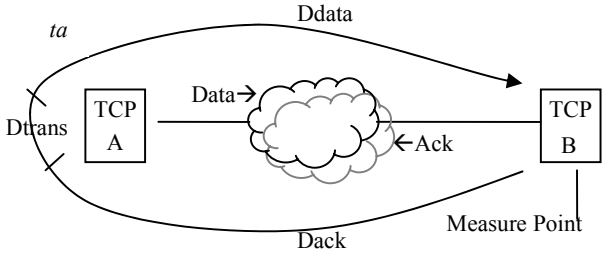

Figure 2 Measurement Scenario

Estimation model includes two models and one criterion:

- TCP sender's non-deterministic finite state transition model: The mathematical model for the TCP sender's internal state transition.
- Channel model: The mathematical model for the *ta* process.
- Maximum-likelihood criterion: The criterion to select the most possible correspondency sequence based on the statistical characteristics of *ta* process.

2.1 TCP Sender's non-deterministic finite state-transition model

In this section, the non-deterministic state transition model that models the sender's internal congestion state transitions is introduced. Before it, a deterministic model is discussed.

[1] The principle of the estimation applies no matter where the dump trace is recorded, at the receiver side or in the middle of the connection. For simplicity, the paper only considers the first case.

There are a set of variables in the TCP sender, such as *cwnd*, *ssthresh*, *dupacks* etc on. The state variables together with the TCP congestion algorithm control the sender's data sending behavior. Let S denote the set of state variables, and S_k denote the state after receiving the kth ack packet. The arrival of A_k causes the sender to send data packet D_j and to shift its state $S_{k-1} \rightarrow S_k$. Ψ is the sender's state transition path:

$$\Psi=\{S_1, S_2, \dots, S_k, S_{k+1}, \dots S_N\}.$$

For a particular TCP implementation, the state transition of Ψ is deterministic and depends only on S_{k-1} and A_k. Therefore, a particular TCP sender can be modeled as a deterministic finite state machine(FSM). A_k and D_j are the input and output of the FSM respectively. Ψ is invisible from the outside.

R_i represents a correspondency sequence constructed from the trace. The first step in the estimation is to eliminate those invalid R_i which do not behave like a TCP. The way to check whether a R_i is a valid TCP correspondency sequence or not is to check whether we can find a valid internal TCP state-transition path Ψ_i for the R_i. If such Ψ_i exists, R_i is a valid TCP correspondency sequence. There are mainly 3 challenges in this checking process:

1. Different TCP implementations have different state-transition rules.
2. A TCP sender has some tunable parameters such as initial window size, sender buffer, which are also very critical in affecting its behavior. However, these parameters are not explicitly reflected in trace, and are difficult to obtain without accessing the sender.
3. When the trace is collected at the receiver side, the data packets sent by the sender may get lost or out-of-ordered on its way to the receiver, so actually we do not know the exact output of the sender. The same thing happens to the acks. We do not know the exact input to the sender too.

There are two approaches to perform this check: The first one is to construct a TCP FSM for each TCP implementation, and then test R_i against every FSM in order to check if it is a valid correspondency sequence of a certain FSM. The second one is to construct a model **M** that depicts the generic behaviors of TCP sender's internal state transitions, and test R_i against this model to check if it violates the generic TCP behavior. The first approach has been used in [13]. Here, we adopt the second approach. Below are two design issues of **M**:

1. R_i violates the generic TCP behavior even when it only violates a part of it. Therefore, **M** currently only depicts generic actions of TCP window increase and state transitions between window increase phase and loss recovery phase. If R_i violates them, it can be safely eliminated because it violates part of TCP behavior. It is relatively easier to define generic actions for them than to define generic actions for TCP "loss recovery", because major differences between TCPs are in their loss recovery actions. When the loss rate is small, a TCP connection is controlled by its window increase action in most of the time. So, **M** can still effectively eliminate impossible correspondency sequences without checking if R_i violates TCP loss recovery actions.
2. State transition inside a particular TCP sender is deterministic because everything is known, the algorithm and the state variables. But when we try to deduce the sender's internal state transition path from outside, situation is different. Here, we know neither the algorithm nor the state variable exactly. Because of this lack of information, when we try to model the state transition for a generic TCP, we must allow multiple different state transitions from a same state upon same input, that is to say that the model **M** is not a deterministic model, but a non-deterministic finite state transition model. In fact, the previous three challenges of the checking process are all intrinsically non-deterministic issues. With this non-deterministic state transition characteristic, different congestion control algorithms used by different TCPs, the implicit sender parameters, and the inexact knowledge of sender's output can be solved.

The rules of **M** represent generic TCP actions. After checking every possible R_i against **M**, we will have a set of possible state transition paths $\Theta=\{\Psi_i\}$. Every path in Θ obeys generic TCP actions and behaves like a TCP. At current stage, we have not determined the correspondency yet, but we have identified all possible correspondency sequences that act like a TCP.

We now briefly describe **M** here. In the model, state S has some basic elements such as θ, *cwnd*, *ssthresh, acks*, where θ denotes the phase of the connection and *acks* denotes the number of new acks received under current *cwnd*. θ controls the general TCP behavior. $\theta \in$ {Slow Start(SS), Additive Increase(AI), Loss Recovery(LR), TimeOut(TO)}. According to different θ values, the sender's states are classified into 4 subsets. Figure 3 shows transitions among them.

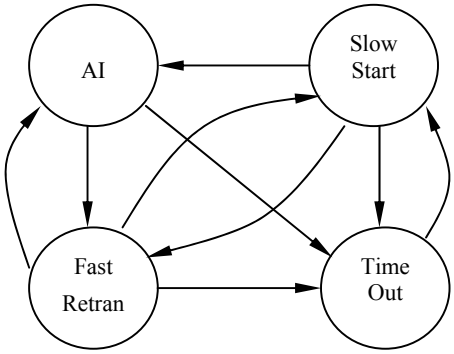

Figure 3 State transitions between subsets of M

The non-deterministic model **M** models the sender's internal state transition behavior by allowing multiple possible state transitions from a same state upon a same input in order to solve the problem caused by the inexact knowledge of the sender's algorithm, sender's implicit parameters, and sender's output. This is illustrated in Figure 4. This graph shows partial state transitions of **M** when *cwnd* ≤ 4. Circles in Figure 4 represent SS states; squares represent AI states; the numbers in them denote the *cwnd* value. Two dashed ellipses are the LR subset and TO subset. Transitions from LR to SS and AI are not drawn in order to keep the figure clear. In this graph, state 3 is a AI state. Usually 3 will shift to 4 when the sender receives 3 new acknowledge packets, otherwise it stays in 3. But some TCP implementations may require that the sender receive 4 new acks for 3 to shift to

④. **M** solves this problem by allowing ③ to generate two new states, ③ and ④, when receiving 3 new ack packets. Other problems such as the implicit sender buffer are solved in a similar way.

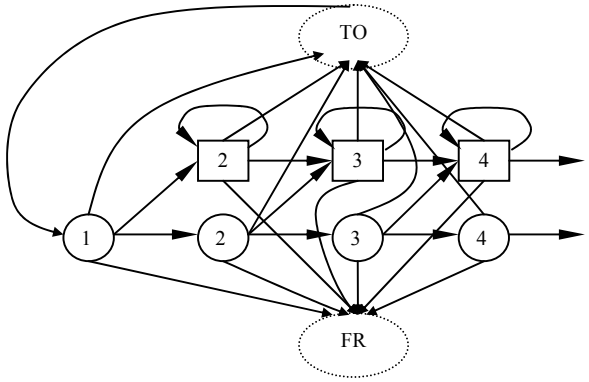

Figure 4 Sketch map of state transitions in M when *cwnd* ≤ 4

2.2 Channel Model

Data packet D_j is triggered by ack A_k. The receiver sent A_k at TA_k, and D_j arrives at the receiver at TD_j, the delay $ta'_j = TD_j - TA_k$. ta'_j is composed of three delays, the *network delay*($Dack_k$) of A_k from receiver to sender, the *server transaction delay*($Dtrans_j$) between the arrival of A_k and the departure of D_j, and the *network delay*($Ddata_j$) of D_j from sender to receiver:

$$ta'_j = Dack_k + Dtrans_j + Ddata_j$$

We ignore *server transaction delay*, because it is usually much smaller than the other two:

$$ta'_j \cong Dack_k + Ddata_j . \quad (1)$$

Channel model is used to describe the statistical characteristics of ta'_j sequence. Now consider two continuous-time stochastic processes: Let $D_a(t)$ denote the *virtual delay* from receiver to sender of an ack packet sent at time t (similar to the concept of *virtual waiting time* in [8]). Let $D_d(t)$ denote the *virtual delay* from sender to receiver of a data packet sent at t, then (1) can be rewritten as:

$$ta'_j = D_a(TA_k) + D_d(TA_k + D_a(TA_k) + Dtrans_j).$$

Sequence $\{ta'_j \mid j = 0, 1, \ldots, l-1, l\}$ is a stochastic process, it is the sum of values of $D_a(t)$ and $D_d(t)$ sampled at time TA_k and $(TA_k+Dack(TA_k)+Dtrans_j)$ respectively.

Here, we suppose $\{ta'_j\}$ has a simple form:

$$ta'_j = m + \eta_j , \quad (2)$$

where m is the mean value, η_j is independent identical random variable with mean 0 and standard deviation σ. In strict sense, using (2) to model $\{ta'_j\}$ process is not correct. Correlations between adjacent ta'_j cannot be reflected in (2). The reason to adopt (2) is because based on this model, we can derive recursive estimation algorithms (3)(4) using *predictive error identification* estimation method in [10]. (3)(4) are similar to the computations of SRTT and VARRTT, which are proposed in [7] and are widely used in TCP sender to dynamically estimate RTT of the connection.[2]

$$m_j = (1-\alpha_1)m_{j-1} + ta'_j \quad (3)$$

$$\sigma_j = \sigma_{j-1} + \alpha_2(|ta'_j - m_{j-1}| - \sigma_{j-1}) \quad (4)$$

Due to the strong correlation between ta values of TCP back-to-back data packets, the channel model actually models another $\{ta_k\}$ sequence:

$$\{ta_k, k = 0, 1, \ldots, N-1, N\}$$

where k is the kth ack packet, ta_k is the average of all ta'_j of data packets D_j trigger by A_k, s

$$ta_k = \frac{1}{J+1}\sum_{l=0}^{J} ta_{j+l}' .$$

The model of $\{ta_k\}$ is

$$ta_k = m + \eta_k , \quad (5)$$

and the recursive estimation algorithms for (m, σ) are

$$m_k = (1-\alpha_1)m_{k-1} + ta_k \quad (6)$$

$$\sigma_k = \sigma_{k-1} + \alpha_2(|ta_k - m_{k-1}| - \sigma_{k-1}) \quad (7)$$

α_1, α_2 are the step sizes. According to the changing rate of the channel, their values are usually between 1/8~1/4.

2.3 Maximum-likelihood Estimation

In 2.1, by comparing the trace to **M**, we have a set Θ. In 2.2, we have a channel model. In this section, we discuss how to select a Ψ_i in Θ that is most likely to be the actual path Ψ.

For every possible transition path $\Psi_i = \{S_1^i, S_2^i, \ldots, S_k^i, S_{k+1}^i, \ldots S_N^i\}$, each S_k^i determines a corresponding pair r_k^i, which determines a ta_k^i. The $\{ta\}_i$ sequence for path Ψ_i is

$$\{ta\}_i = \{ta_1^i, ta_2^i, \ldots ta_k^i, ta_{k+1}^i, \ldots ta_N^i\}.$$

We can calculate the likelihood value $P(\{ta\}_i)$ for every sequence $\{ta\}_i$, if we know the probability density function of ta_k—$f_k(t)$. From the aspect of satisfying TCP rules, every path Ψ_i in Θ is equally possible, we cannot judge which one is more possible. However, the likelihood values $P(\{ta\}_i)$ of these paths are different. Larger likelihood value $P(\{ta\}_i)$ means larger possibility in sense of ta process. So, based on certain channel model assumption, we select the $\{ta\}_i$ with the largest likelihood value and its counterpart Ψ_i as our estimation result:

$$\psi_i = \arg\max_{\{\psi_i\}} P(\{ta\}_i) ,$$

for every possible $\psi_i \in \Theta$.

According to channel model (5), ta_k are independent from each other, so,

$$P(\{ta\}_i) = f(ta_1^i) \times f(ta_2^i) \times \cdots \times f(ta_N^i)$$

[2] The author in [9] had a different explanation of SRTT algorithm.

$$L(\{ta\}_i) = \log P(\{ta\}_i) = \sum_{k=1}^{N} \log f(ta_k^i) = \sum_{k=1}^{N} \lambda(ta_k^i)$$

$$L(S_k^i) = L(S_{k-1}^i) + \lambda(ta_k^i)$$

If (m, σ) is known and ta_k is normal distribution, we have:

$$\lambda(ta_k^i) = -\frac{1}{2\sigma}(ta_k^i - m)^2 - \log 2\pi\sigma \quad (8)$$

$$L(\{ta\}_i) = \sum_{k=1}^{N} -\frac{1}{2\sigma}(ta_k^i - m)^2 - \log 2\pi\sigma \quad (9)$$

$$L(S_k^i) = L(S_{k-1}^i) - \frac{1}{2\sigma}(ta_k^i - m)^2 - \log 2\pi\sigma \quad (10)$$

The rationale of the Maximum-likelihood criterion used in selecting Ψ_i is that if a Ψ_i deviates from actual Ψ, its $\{ta\}_i$ will also deviate from actual $\{ta\}$, which makes $\{ta\}_i$ deviate from the model of *ta* process. Thus its likelihood value is very likely to be smaller than the value of actual $\{ta\}$. The more it deviates from the actual, the smaller the likelihood value it has. Therefore, based on this criterion, the selected path will be at least very close to the actual path even if it is not the actual one.

Maximum-likelihood Estimation(MLE) is widely used in many fields such as parameter estimation, convolution-code decoding, and channel equalization [16]. For the first one, the maximum-likelihood parameter estimation have already found its application in the network tomography[5]. In the latter two fields, MLE is used to select a most possible path among many candidate paths. The most possible path is the path with maximum-likelihood value. The uses of MLE in these two fields are very similar to its use here.

3. ESTIMATION METHOD

3.1 Use M to search possible path set Θ

The TCP sender's non-deterministic FSM $\mathbf{M} = (K, \Sigma, \Delta, s, F)$, where K is the set of finite states $\{q_0, q_1, \ldots q_n\}$, Σ is the accepted input. Δ is the transition function, $K \times \Sigma \rightarrow K$, $s \in K$ is the initial states, F is the stopping state. The accepted input Σ is the corresponding pairs r_k. r_k is generated based on the trace. r_k must satisfy time causality.

$k = 0$:$S_0 = s$, $\Omega_0 = \{S_0\}$
step k: for $k = 1$ to $N - 1$
  for every S_{k-1}^i in Ω_{k-1}
    create new possible r_k^i
    if $S_{k-1}^i \times r_k^i \rightarrow S_k \in K$
      store S_k in Ω_k
    end
  end
end
OUTPUT: $\Omega_N = \{S_N^i\}$, $\Theta = \{\Psi_i\}$

S_{k-1}^i is the ith possible state in Ω_{k-1}, Ω_{k-1} is the set of all S_{k-1}.

The searching procedure is recursive. At every step, possible r_k is created for S_{k-1}^i and inputted into **M**. If r_k is accepted by **M**, then create S_k, otherwise discard it. Multiple S_k may be created for each S_{k-1}^i. Repeat the above step until reach the last ack A_N. In the end, we get Ω_N and Θ. Above is the pseudo-code of the recursive algorithm.

3.2 Adaptive Maximum-likelihood Estimation

For every new created S_k^i, its likelihood value is computed using (10),

$$L(S_k^i) = L(S_{k-1}^i) - \frac{1}{2\sigma}(ta_k - m)^2 - \log 2\pi\sigma \ ,$$

where S_{k-1}^i is the predecessor of S_k^i.

However, in reality, we usually do not priori know the channel coefficients (m, σ) before estimation. Although sometimes (m, σ) can be obtained by *ping*, they may not be applied to a TCP because of the different sampling methods used by *ping* and TCP. Moreover, if the channel is time-varying, we must be able to track the change of the channel coefficients.

The principle of adaptive maximum-likelihood estimation is to use the tentative decision to update the channel estimation during estimation process. The channel estimator block(see Figure 5) uses (11) to track the channel changes according the tentative output of the Viterbi block(explained in 3.3).

$$(\mu,\sigma)_k = F((\mu,\sigma)_{k-1}, ta_k) \quad (11)$$

F is the update function. Here, we use (6)(7) as F to adjust channel estimation.

In each step of the recursive algorithm, there are many new states $\Omega_k = \{S_k^i\}$, in which at most there is only one correct S_k^i. Therefore, the choice of ta_k in (11) is a problem. Generally, there are two approaches to solve this problem. We briefly discuss them below, whereas further discussion of this problem is beyond the scope of this paper:

1. In each step, we have only one copy of channel estimation (μ,σ). The estimation has explicit physical explanation. It is the estimation of actual channel. In principle, the closer the estimated channel coefficients to the actual value, the better performance of the method. According to this principle, ideally ta_k used in (11) should be the value of the actual r_k. One reasonable method is to choose ta_k according the likelihood value of S_k, i.e. choose

$$ta_k = \arg\max_{\{S_k^i\}} \{L(S_k^i)\}$$

to update $(\mu,\sigma)_{k-1}$. However, this method has its limitation, state S_k^i with maximum L does not guarantee that S_k^i is the actual state, that its ta_k^i is the actual *ta*, so as not to guarantee that the updated $(\mu,\sigma)_k$ reflect the actual state/parameters of the channel. The deviation between the estimated channel parameters and the actual channel parameters may lead to next wrong choice of *ta* in (11), and may further lead to larger deviation between the estimation and the actual. The deviation may propagate and finally lead to serious estimation error.

2. Per-Surviving Processing(PSP): In each step, every surviving state S_k^i maintains its own channel estimation $(\mu,\sigma)_k^i$ which is calculated using its own ta_k^i estimation result by (12). Also, its likelihood value $\lambda(S_k^i)$ is calculated using its own

channel estimation by (13). This approach is used in solving ISI in MSLE by Rehali et al in [17], and Sechadri in [18].

$$(\mu,\sigma)_k^i = F((\mu,\sigma)_{k-1}^i, ta_k^i) \quad (12)$$

$$\lambda(ta_k^i) = -\frac{1}{2\sigma}(ta_k^i - m_k^i)^2 - \log 2\pi\sigma_k^i \quad (13)$$

3.3 Viterbi Algorithm

In each step, every S_{k-1} may generate multiple new states S_k, the number of states in Ω_k will grow exponentially with k. Because of the Markov property of the sender state transition, Viterbi algorithm can be used to reduce the number of states in Ω_k effectively.

Viterbi algorithm was proposed as a method of decoding convolution codes at first. Since that time, it has been recognized as an attractive solution to a variety of digital estimation problems. The rationale of Viterbi algorithm is to use two properties of the estimated object—the Markov property and the finite property—to discard impossible path as soon as possible. This reduces the complexity in searching[16]. The pseudo-code of the Viterbi algorithm is written as:

$\hat{S}_k$ = arg max $L(S_k)$, for all S_k in Ω_k of same state
store $\hat{S}_k$ and delete the others

3.4 Description of Estimation Procedure

Figure 5 and Figure 6 show the block-diagram and the pseudo-code of the method. In *tcpdep*, we adopt PSP in searching ML path.

3.5 Rules of M

Rules of the non-deterministic TCP state-transition machine **M** is constructed based on the basic algorithms stipulated in RFC2581[3] and specifications in some other RFC documents[1, 2, 6, 11, 15].

There are mainly two aspects of considerations in constructing the state-transition rules:

- Strict rules reduce the maximum likelihood searching space Θ but may exclude some valid state-transition paths from Θ.
- Loose rules expand the maximum likelihood searching space Θ but may not achieve the goal of effectively eliminating invalid state transition paths out of Θ.

Therefore, the rules of **M** are the result of trade-off between strict rules and loose rules. The principle is to eliminate invalid state transition paths as many as possible on the condition of ensuring that the correct path will not be eliminated at this stage.

Rules of **M** is of two categories: The first category depicts the transition rules between four state subsets showed in Figure 3. The second category depicts transitions rules in each subset. Details of rules are not discussed in this paper.

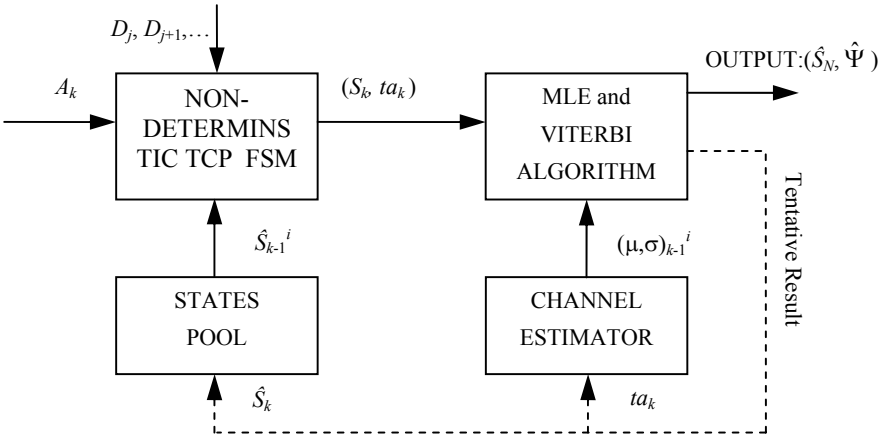

Figure 5 Block diagram of the estimation procedure

$k = 0$: $S_0 = s$, $\Omega_0 = \{S_0\}$

step k: for $k = 1$ to N -1

for every $\hat{S}_{k-1}^{\,i}$ in Ω_{k-1}

create new possible $r_k^{\,i}$

if $\hat{S}_{k-1}^{\,i} \times r_k^{\,i} \rightarrow S_k \in K$

store S_k in Ω_k

end

end

for every new state $S_k^{\,i}$ in Ω_k, i for the ith state

$L(S_k^{\,i}) = L(S_{k-1}^{\,i}) + \lambda(ta_k^{\,i})$

$(\mu,\sigma)_k^{\,i} = F((\mu,\sigma)_{k-1}^{\,i}, ta_k^{\,i})$

end

$\hat{S}_k$ = arg max $L(S_k)$, for all S_k in Ω_k of same state

store $\hat{S}_k$ into Ω_k and delete the others

end

OUTPUT: $(\hat{S}_N, \hat{\Psi})$ = arg max $L(S_N)$, for all S_N in Ω_N

Figure 6 Pseudo-code of the estimation procedure

4. VALIDATION

As mentioned before, the method cannot guarantee the estimated result match the actual exactly. In this section, we perform both simulations in NS and Internet experiments to test the estimation performance.

In simulation, we study the estimation performance under different kinds of network background traffic and with various TCP sender versions. The sender always have data waiting to send, thus its data sending behavior is only controlled by the TCP congestion control mechanisms.

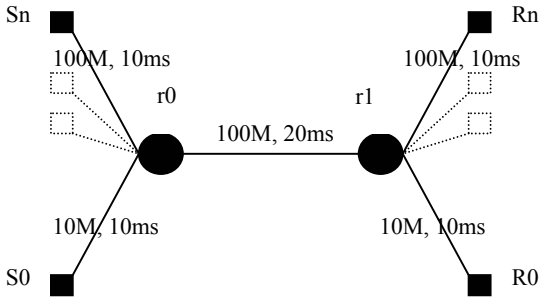

Figure 7 Simulation topology--dumbbell

From Section 4.1 to 4.8 we use dumbbell topology showed in Figure 7. The measured TCP connection is between S0 and R0. S0 is the sender. R0 is the receiver. Validation procedure is as follows: First, when simulation runs, we measure at both S0 and R0 and dump the traffic into two different files. When simulation stops, we use *tcpdep* to estimate the sender behavior with the trace dumped at R0. Then we analyze both traces dumped at S0 and R0 jointly to obtain the actual sender behavior and network delay and loss. Finally, we evaluate the estimation performance by comparing the estimated result and the actual. In all simulations without particular indications, the sender uses TCP *NewReno* version, sends fixed 1600 data packets(excluding retransmit packets), the delay ack is enabled at the receiver. The maximum window size is 40 packets. The fixed propagation round-trip time between S0 and R0 is 80ms. The bandwidth of link r0-r1 is 10Mbps with a buffer size of 160 packets. Therefore, the link queuing delay is between 0~128ms.

In simulation, Sn-Rn share with S0-R0 the same bottleneck r0-r1. Generating different kinds of traffic between Sn-Rn makes the queue of link r0-r1 exhibit different queuing dynamics, which causes the measured TCP connection to experience different network delay and loss processes. This allows us to evaluate the estimation performance under different network conditions. Changing the TCP sender version of S0 allows us to investigate the method's adaptability to different TCP sender versions.

In 4.9, we perform simulation under tandem network topology with cross-traffic. In 4.10, we show the Internet experiment result.

4.1 Performance under Poisson background traffic

In this simulation scenario, the background traffic is Poisson with an average arriving rate λ of 9.6Mbps. Using Poisson traffic does not mean we think the background traffic in the Internet is Poisson. It is only used as a method to introduce random queuing delay. We also adopted other kinds of background traffic in the following sections.

To eliminate randomness effects of background traffic, we repeated 64 runs. The average ε_W of 64 runs is 4.0% and ε_{ta} is

4.4%[3]. Figure 8 shows the 59th run. It represents the typical performance of the 64 runs. In this run, ε_W is 3.6% and ε_{ta} is 4.5%, which approximate to the average values. From this graph, we can see that the estimated result matches the actual result very well in most of the time. The errors mainly occur in the loss recovery phase. One reason may be that currently we allow arbitrary pairing combinations between acks and data packets during loss recovery phase. However, it is not the intrinsic drawback of our method. The errors will be reduced with the improvement of **M**. Viewing the errors from another prospective, even though serious estimation errors occur in the loss recovery phase, the errors do not propagate, which is a good property.

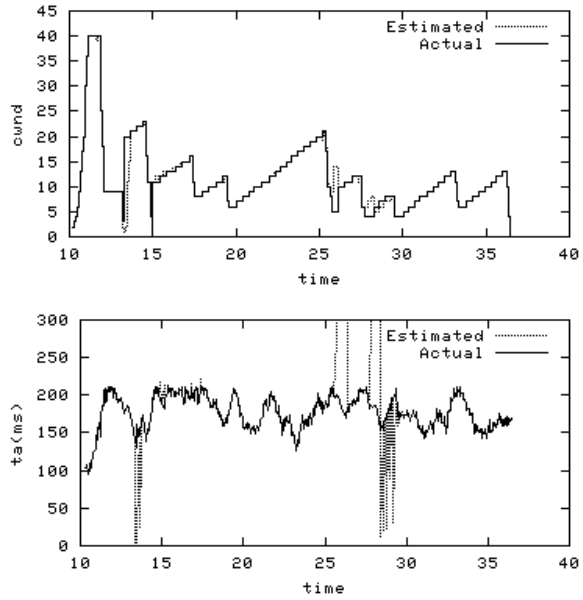

Figure 8 Comparison of congestion window process and *ta* process between estimation and actual for the 59th run

Changing the λ of Poisson background traffic changes the loss rate experienced by the measured TCP. Figure 9 shows the comparison of average *cwnd* and *ta* between estimated and actual results when we vary the λ from 8Mbps to 10.8Mbps. In the graph, we also show e_W and e_{ta} respectively. Totally 64 simulation runs are performed. As λ increases, the loss rate of the measured TCP grows from 0 to 0.053. The average ε_W of 64 runs is 3.9% and ε_{ta} is 2.5%, which are both very small. From this graph, we see that the e_W and e_{ta} for every individual run is small too. The figure shows the good estimation performance of the method under different loss rates in this scenario.

4.2 Performance under long-duration background TCP traffic

Different kinds of background traffic have different impacts on the queuing dynamics, which causes the measured TCP to experience different *ta* processes and affects the estimation performance. In last section, the background traffic is Poisson. In the following two sections, we will use two other different kinds of background traffic.

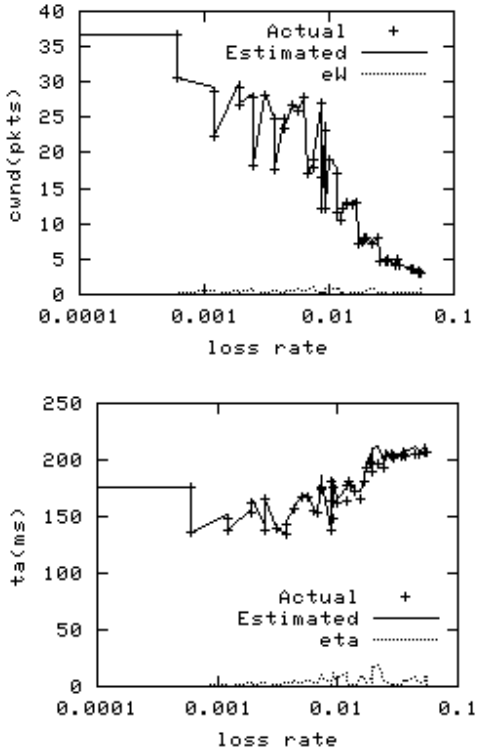

Figure 9 Comparison of average *cwnd* and *ta* between estimated and actual result under Poisson traffic

In this section, we use long-duration TCP as background traffic(FTP traffic). Figure 10 shows the result when we vary the number of background TCP flows from 5 to 160. There are totally 64 runs. The loss rate of the measured TCP grows from 0 to 0.05 as the number of background flows increases. The average ε_W of 64 runs is 2.6% and ε_{ta} is 1.0%. From Figure 10, we see that the e_W and e_{ta} for each individual run is small too. The estimation performance under this background traffic is also very good.

4.3 Performance under FTP+HTTP background traffic

In this section, we introduce a more realistic background traffic, which is FTP + HTTP traffic. Long-duration FTP traffic is of 10 flows. HTTP traffic is generated using *webcache* module in NS. In the simulation, we vary the number of *active*[4] HTTP flows over the link r0-r1 to change its congestion level.

There are totally 64 runs. The loss rate of the measured TCP is from 0.001 to 0.056. The average ε_W of 64 runs is 3.5% and ε_{ta} is 2.7%. Figure 11 shows the result for each run.

[3] Definitions of estimation performance metrics e_W, e_{ta}, ε_W, ε_{ta} are in Appendix A.

[4] A slight change is made in NS webcache module to control the number of *active* HTTP flows.

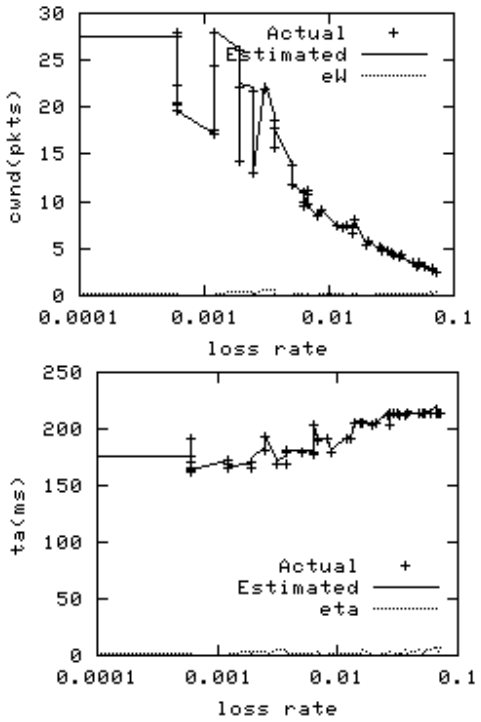

Figure 10 Comparison of average *cwnd* and *ta* between estimated and actual result under FTP traffic

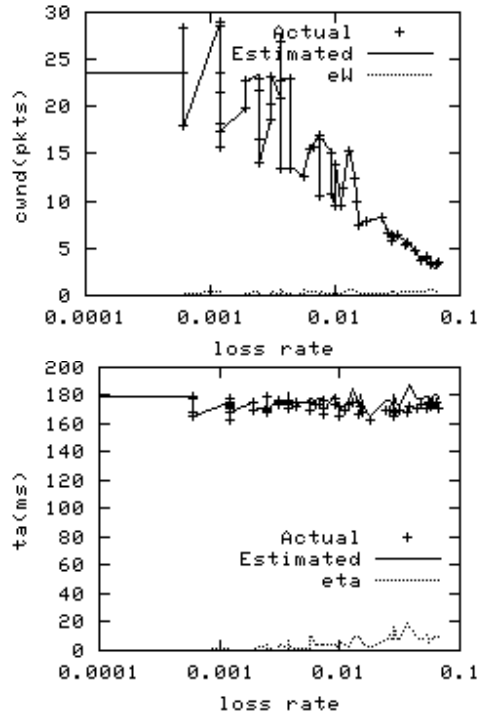

Figure 11 Comparison of average *cwnd* and *ta* between estimated and actual result under FTP+HTTP traffic

4.4 Effects of different background traffic

In previous three sections, we have tested our method under three kinds of background traffic. In all cases, the method exhibits good performance under different data loss rates. In this section, we study the effects of different kinds of background traffic on *ta* process and on the estimation performance.

We compare the estimation performance under three different background traffic, the 9.6Mbps Poisson traffic, the 65 FTP flows, and the 10FTP + 180HTTP mixed traffic. We have 64 runs under each of the three. In three simulation scenarios, the average loss rates(p) of the measured TCP are all approximately 1.2%.

Table 1 Average result of each kind of traffic

	p	W	$ta(ms)$	$ta\_std(ms)$	ε_W	ε_{ta}
Poisson	1.2%	12.4	175	27.3	4.0%	4.4%
FTP	1.1%	9.06	195	20.4	1.5%	1.2%
FTP+HTTP	1.1%	11.9	170	26.5	3.3%	3.9%

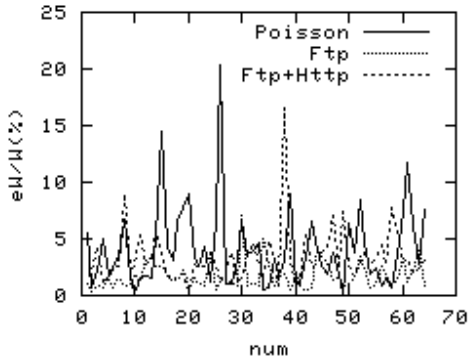

Figure 12 ε_W under different background traffic

Table 1 shows the average performance. Figure 12 shows the e_W for every run. *num* in the figure represents the nth run. The e_{ta} for every run is similar to the e_W, thus not showed. From them, we see that the performance under pure FTP background traffic is the best among the three, the average ε_W and ε_{ta} of 64 runs is between 1%~2%. The performances under Poisson and mixed FTP+HTTP are similar, in which the average ε_W and ε_{ta} are both around 4%.

To explained the difference, we plot Figure 13 to show the *ta* processes experienced by the measured TCP under different background traffic. The *ta* processes are visually quite different. From Figure 13 and Table 1, we see that the variance of *ta* process is the smallest for the FTP traffic, while almost the same for the other two kinds of traffic. We believe this explains why ε_W and ε_{ta} are smaller under FTP background traffic than under the other two—smaller *ta* variance leads to better estimation performance. However, although *ta* processes under Poisson and mixed traffic are visually very different(*ta* of the former varies slowly while *ta* of the latter varies more abruptly), their performances are alike.

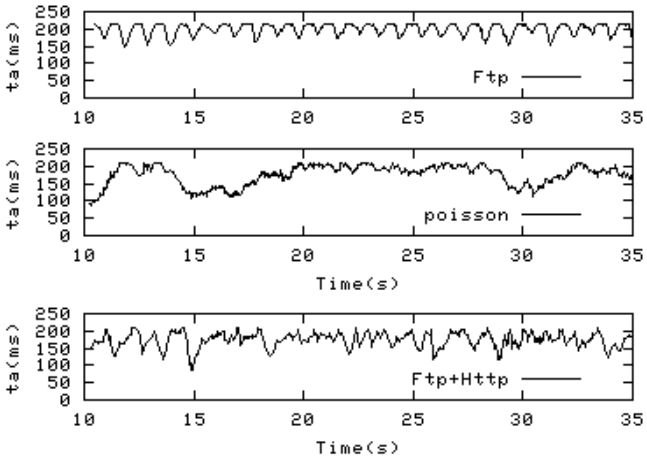

Figure 13 *ta* processes under different background traffic, FTP, Poisson and FTP+HTTP

4.5 Effects of queue length

Increasing queue length of link r0-r1 gives the *ta* of the measured TCP a larger fluctuation range, which might reduce the estimation performance. To explore this issue, we consider 3 scenarios under FTP+HTTP background traffic where queue lengths of link r0-r1 are 160, 320 and 640 respectively. In each scenario, we vary the number of active HTTP flows to change the loss rate of the link. For each number of flows, we have 8 runs. Figure 14 shows the effects of queue length on the ε_W under various loss rates. The graph of ε_{ta} is similar. In the graph, each point represents the average loss rate and ε_W of 8 runs under the same number of flows. From this graph, we cannot see the effects of queue length on the performance. The reason is probably because that the variance of *ta* process does not grow linearly with the queue length as we assumed. (see Figure 15).

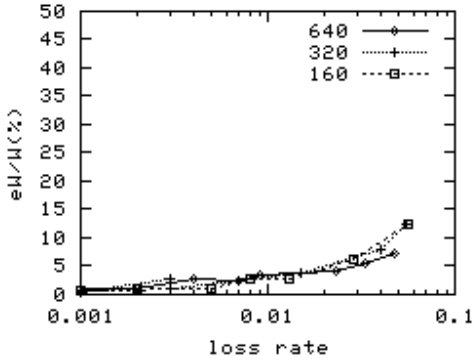

Figure 14 Effects of queue length on the estimation performance

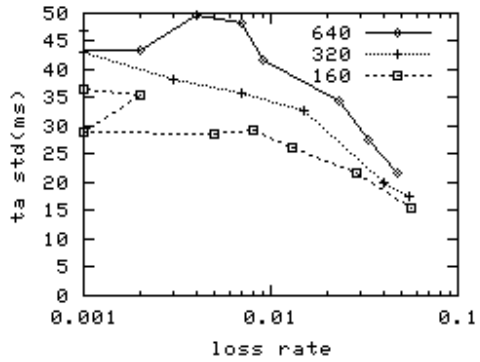

Figure 15 Effects of queue length on the standard deviation of *ta* process

4.6 Performance of different TCP versions

One advantage of the method is that we can estimate TCP senders' behavior of different versions using a generic TCP sender model. In this section, we evaluate the performance for 5 different TCP versions in NS. The 5 versions are *tahoe*, *reno*, *newreno*, *Sack1* and *Fack*. We had 16 runs for each version. The background traffic is Poisson—9.6Mbps. The average loss rate is 1.2%. Table 2 shows the average performance for each version. From the table, we see that our method achieves similar good performance for TCP *tahoe*, *newreno*, *Sack*1 and *Fack*, while the performance for TCP *reno* is worse than the other four. Figure 16 shows the ε_W performance for every run. ε_{ta} is similar to ε_W.

Table 2 Average ε_W, ε_{ta} for different TCP versions

	tahoe	Reno	newreno	Sack1	Fack
ε_W	1.3%	13.3%	3.6%	1.4%	3.9%
ε_{ta}	1.7%	13.6%	3.9%	1.4%	3.2%

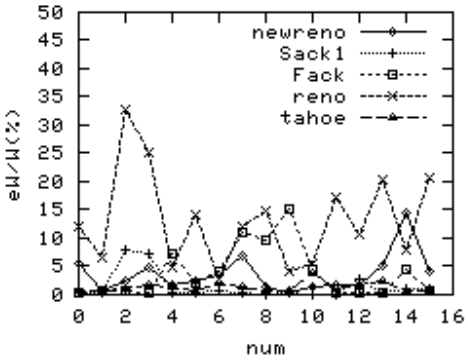

Figure 16 ε_W and ε_{ta} for every run

4.7 Performance under ACK loss

If ack A_k is lost, the data packets which should be triggered by A_k are now triggered by A_{k+1}. If the time interval between A_k and A_{k+1} is small when they are sent from receiver, it is difficult for us to discern whether A_k is lost or not either from the rule or from *ta* value. In this situation, regarding that A_k is not lost, which equals

to regard that the data packets actually triggered by A_{k+1} are triggered by A_k, will not affect the path's likelihood value very much. The wrong decision will be confined to only those data packets which should have been triggered by A_k, which is a quite limited error event.

If the time interval is large, to regard that the data packets actually triggered by A_{k+1} are triggered by A_k will cause a sudden increase of *ta* value. If the increase exceeds certain threshold, we can judge that A_k is lost. Therefore, ack loss will still not affect the method in this situation.

To investigate the effects of ack loss, we introduce ack loss in the measured TCP in simulation. The ack loss rates are 0, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1 and 0.2. We use 9.6Mbps Poisson background traffic, and perform 64 runs for each loss rate. Figure 17 shows the result. From it, we see that when the ack loss rate is less than 0.1, the performance nearly stays in the level where ε_W and ε_{ta} are around 5%. When loss rate exceeds 0.1, the performance degradation becomes obvious. When loss rate is 0.2, ε_W and ε_{ta} are 15%.

Figure 18 shows the every run for simulation scenarios of no ack loss and of 0.01 ack loss rate. The average ε_W, ε_{ta} of the former is 4.0% and 4.4%, and 4.7% and 5.9% of the latter. We can see from the graph that the performance of the latter is only slightly worse than the former.

From these two graphs, we conclude that when the ack loss rate is small, the performance degradation is tolerable. For larger ack loss rate, the method needs to be improved.

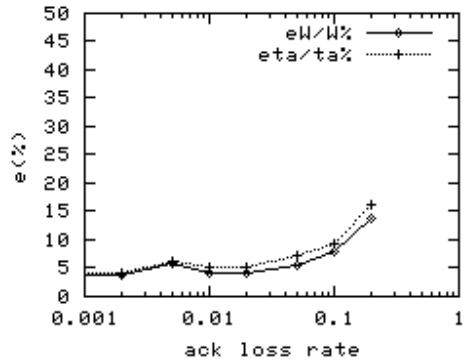

Figure 17 Average ε_W, ε_{ta} with ack loss from 0 to 0.2

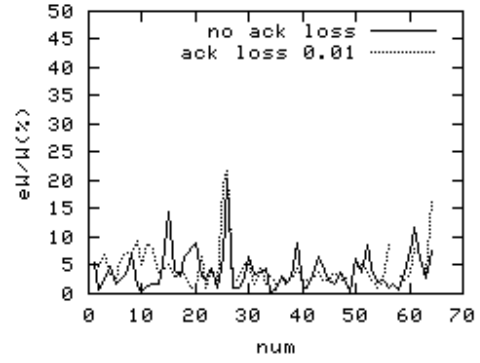

Figure 18 Comparison between no ack loss and 0.01 ack loss

4.8 Performance under no background traffic

In previous experiments, the traffic of the measured TCP is not the main traffic in the network. In this section, we consider a simple scenario where the TCP is the only traffic in the network. In this situation, the *ta* process it experiences has strong correlation with its data sending behavior. With the increase of TCP congestion window, the number of packets in the network also increases. When the number exceeds a certain threshold, packets in router's buffer begin to increase steadily as the window increases. The increase continues until loss occurs. Then, the sender suspends sending data. The router's buffer begins to decrease. When the sender resumes sending data packets, another cycle of window increase begins. In the cycle, *ta* process increases slowly at first, and then deceases suddenly. This experiment also represents a typical scenario where the delay changes dramatically.

Consider Figure 7 network topology. When the queue of r0-r1 increases one more packet, *ta* = *ta* + Packetsize / BW. The smaller BW, the bigger *ta* increment for every one more packet. The maximum queue length determines the maximum value of *ta*. The larger queue length, the larger maximum *ta* value, and the larger *ta* drop after loss occurs.

We consider a simulation scenario where BW = 64kbps. This equals to the bandwidth of a Internet dail user, and it should almost be the worst case we can meet in the Internet. The fixed RTT is 80ms. In simulation, we change the queue length from 5 to 40. The TCP window grows linearly from 1 packet. When the queue length is 40, the data packets will not lost, because the maximum TCP window size is 40.

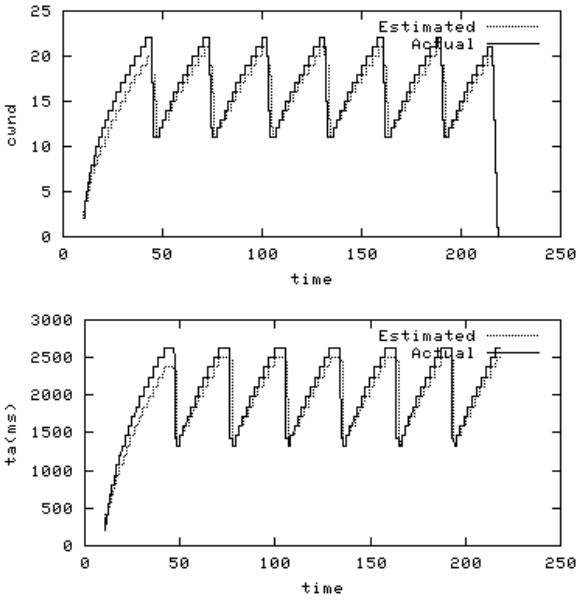

Figure 19 Performance when queue length is 20 packets

Figure 19 is the comparison between the estimation and the actual when the queue length is 20. The actual average *W* is 16.4, and the average estimated *W* is 15.8. ε_W is 7%. The average actual *ta* is 2025ms, and the estimated *ta* is 1957ms. ε_{ta} is 7%.

Figure 20 shows the ε_W, ε_{ta} under different queue lengths, all ε are less than 12%. The average ε_W of 8 runs is 7%, ε_{ta} is 7.5%. When the queue length is 40, ε_W and ε_{ta} are both 2.2%. Thus, we

conclude that when the estimated TCP is the only traffic in the network, the method can track the change of *ta* with some lag.

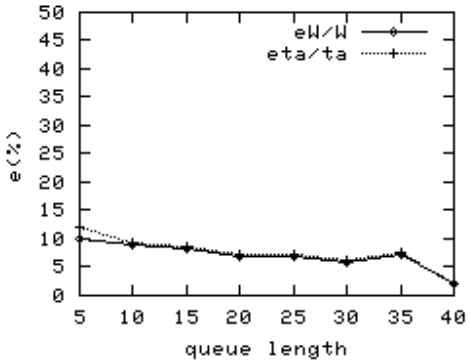

Figure 20 ε_W, ε_{ta} under different buffer size

4.9 Simulation using Tandem Network Topology

In this section, we use a network topology showed in Figure 21. The measured TCP is between S0 and R0, and we generate cross traffic from Si to Ri (i = 1, 2, … n). We also generate traffic from Sn+1 to Rn+1. In this simulation scenario, we analyze the estimation performance under multiple congested links and two-way traffic. The traffic between each pair of Sn and Rn are composed of 8 long-duration TCP flows and 100 active HTTP flows. Traffic between different pairs are independent. In the experiment, we increase the number of congested links from 1 to 10 in order to make RTT between S0 and R0 more variable. We repeat 64 times for each number of links. From Table 3, we see that the variance of *ta* increases as the number of links grows. However, ε_{ta} and ε_W do not grow. We think the reason is because that the growth of *ta* average(m_{ta}) counteracts the effect of the growth of *ta* standard deviation(σ_{ta}). In Table 3, we see that the σ_{ta}/m_{ta} decreases as the number of links grows.

In the simulation, we also observed ack compression phenomenon in the measured TCP according to the definition made by Paxson in [14]. However, the magnitude of ack compression is small in this simulation scenario, so we did not observe the obvious sign that ack compression adversely affects the estimation performance here.

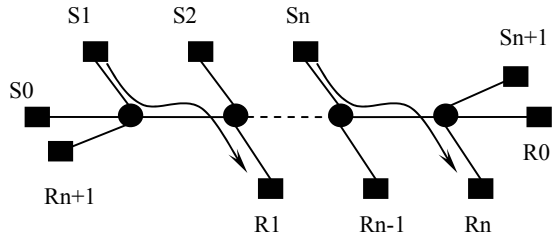

Figure 21 Simulation Topology -- Tandem Network with cross-traffic

Table 3 Simulation Results[5]

Links	1	2	3	4	5	10
m_{ta}	143.9	235.9	332.7	424.8	517.1	1018
σ_{ta}	19.6	27.0	35.2	41.4	44.8	71.4
σ_{ta} / m_{ta}	13.6%	11.4%	10.6%	9.7%	8.7%	7.0%
max-min	98.8	153.4	197.2	247.6	262.4	435.9
W(aver)	21.2	19.9	17.1	15.4	17.8	13.0
ε_{ta}	5.1%	5.4%	5.9%	4.1%	3.7%	3.8%
ε_W	4.2%	4.7%	5.2%	3.8%	3.4%	3.8%

4.10 Internet Experiments

We performed experiments in the Internet. The sender and receiver are in two different ASes. There are 13 hops between the sender and receiver. The RTT varies greatly, from 20ms in the night to 2000ms in the daytime. There are nearly no loss between the two hosts, so near the receiver we placed another FreeBSD host as a router. It uses *dummynet* to introduce losses. The sender is FreeBSD4.5 and uses TCP *newreno*. The receiver issues a file request every 10 minutes. The file is approximately 1.7MB, and needs roughly 1200 data packets(MSS1460).

Figure 22 shows the estimation result of 208 TCP connections recorded at 02-19-2003. The average data loss rate and ack loss rate are 0.02. The average *ta* for each connection is from 10ms to 2000ms. 185 out of 208 connections gave results. 23 connections fail to give results because the correct correspondency sequence is eliminated prematurely during the searching process. The average ε_W of 185 connections is 8.6%, and ε_{ta} is 6.5%. From this graph, we can see that 80% of the connections have ε_W and ε_{ta} less than 10%, 90% of connections have ε_W and ε_{ta} less than 20%. Connections with large ε are those with small RTT(around 20ms) and large RTT variations where the peak RTT can suddenly reach 60ms at a certain period of the connection. Further investigation is needed in this case.

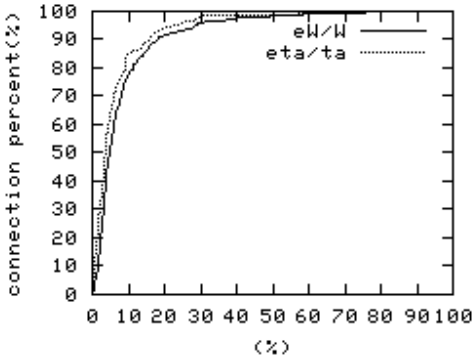

Figure 22 The ε_W, ε_{ta} cumulative probability distribution of 185 TCP connections

[5] max-min is (maximum *ta* – minimum *ta*) in a connection.

5. CONCLUSION AND FUTURE WORK

This paper proposed a method to estimate the correspondency between acks and data packets of a TCP connection. The advantage of the method is that the estimation does not require to use dump trace recorded at or near the sender side. Based on this method, we developed a tool *tcpdep*. It enables us to perform the estimation using *tcpdump* trace passively measured at the receiver side. Simulations and Internet experiments show the feasibility of this method.

Below, we give two prospective uses of the method:

1. Analyze the RTT dynamics of a TCP at the receiver side: In many situations, analyzing the dynamics of round trip time between two points in the network reveals important network path properties between these two points. *ping* is such a tool. However, the RTT dynamics of a TCP connection is tightly associated with its data sending manner. Different data sending manners may result different RTT dynamics. Even tools such as *TReno* are difficult to simulate the exact behavior of a real TCP connection. So, the only way to analyze the RTT dynamics of TCP is to establish a real TCP connection. However, currently such analysis can only be performed with traces recorded at or near the TCP sender. This is because that when only the trace recorded at the receiver is provided, we do not know which data packet is triggered by which ack packet. Our method gives us this crucial correspondency information and enables us to analyze the dynamics of *ta*, which is a kind of RTT in definition. The analysis of *ta* will give us the exact picture of how round-trip time varies during a real TCP connection. The receiver-based analysis technique gives us a lot of freedom.

2. Analyze the interactions between TCP and queuing dynamics in real network environment: Detailed analysis of interactions between queuing dynamics and TCP is usually confined in simulation environment now. One reason is that it is easy for us to dump traffic of end hosts and of the router's queue simultaneously in simulation environment, while it is difficult to do such things in real network environment. If the estimation of this method is accurately enough, we only need to measure at the router, and use the method to estimate the sender behavior.

In the development of *tcpdep*, firstly, we found that to create a generic TCP congestion state-transition model for real TCP implementations is a difficult task. Secondly, we found it hard to judiciously explore the simulation parameter space, such as topology, queue length, link delay, and traffic type when we try to validate the estimation performance.

In the future, careful and thorough studies of real TCP implementations' internal state transition path or their measurable ack and data pairing regulations are important. Moreover, further investigation of the channel model and analysis of the estimation performance are necessary.

6. ACKNOWLEDGMENTS

The authors thank for Xuelong Zhu for his many insightful comments on the work. The authors would like to thank the anonymous reviewers for their comments, which have greatly helped us in preparing this paper.

APPENDIX A—ESTIMATION PERFORMANCE METRICS

There are two kinds of performance metrics: The first metric is to describe the closeness of the estimated *cwnd* process to the actual *cwnd* process. The second one is to describe the closeness of estimated *ta* to the actual *ta* for every packet.

e_W and ε_W are the metrics in describing the difference between estimated *cwnd* process and the actual *cwnd* process:

$$e_W = \frac{\int_0^T |W_0(t) - W(t)|\, dt}{T},$$

$$\varepsilon_W = \frac{e_W}{\overline{W}} = \frac{\int_0^T |W_0(t) - W(t)|\, dt}{\int_0^T W_0(t)dt},$$

where, $W_0(t)$ represents the actual *cwnd* process, and $W(t)$ is the estimated *cwnd* process. T is the total time of the connection. Ideally, e_W and ε_W are 0.

e_{ta} and ε_{ta} are the metrics in describing the difference between actual *ta* sequence and the estimated *ta* sequence:

$$e_{ta} = \frac{\sum_{j=1}^{N} |ta_j^0 - ta_j|}{N},$$

$$\varepsilon_{ta} = \frac{e_{ta}}{\overline{ta}} = \frac{\sum_{j=1}^{N} |ta_j^0 - ta_j|}{\sum_{j=1}^{N} ta_j^0},$$

where ta_j^0 represents the actual *ta* for the *j*th data packet, and ta_j is the estimated *ta* for the *j*th data packet. N is the total number data packets for the TCP connection. Ideally, e_{ta} and ε_{ta} are 0.

7. REFERENCES

[1] Allman, M., H. Balakrishnan, and S. Floyd, *Enhancing TCP's Loss Recovery Using Limited Transmit.* RFC3042, 2001.

[2] Allman, M., S. Floyd, and C. Partridge, *Increasing TCP's Initial Window.* RFC2414, 1998.

[3] Allman, M., V. Paxson, and W. Stevens, *TCP Congestion Control.* RFC2581, 1999.

[4] Barford, P. and M. Crovella. *Critical Path Analysis of TCP Transactions*. in *SIGCOMM '00*. 2000.

[5] Caceres, R., et al., *Multicast-Based Inference of Network Internal Loss Characteristics.* IEEE Trans. ON Information Theory, 1999. **45**(7): p. 2462-2480.

[6] Floyd, S. and T. Henderson, *the Newreno Modification to TCP's Fast Recovery Algorithm.* RFC2582, 1999.

[7] Jacobson, V. *Congestion avoidance and control*. in *SIGCOMM '88*. 1988. Stanford, CA,.

[8] Kleinrock, L., *Queueing Systems*. 1976: Wiley, NY.

[9] Li, Q. and D.L. Mills, *Jitter-Based Delay-Boundary Prediction of Wide-Area Networks.* IEEE Trans. on Networking, 2001. **9**(5): p. 578-590.

[10] Ljung, L. and T. Soderstrom, *Theory and Practive of Recursive Identification*. 1983: The MIT Press.

[11] Mathis, M., et al., *TCP Selective Acknowledgment Options*. RFC2018, 1996.
[12] Padhye, J. and S. Floyd. *On Inferring TCP Behavior*. in *SIGCOMM'01*. 2001.
[13] Paxson, V. *Automated Packet Trace analysis of TCP Implementations*. in *SIGCOMM '97*. 1997.
[14] Paxson, V., *End-to-End Internet Packet Dynamics*. IEEE Trans. on Networking, 1997. **7**(3): p. 277-292.
[15] Paxson, V. and M. Allman, *Computing TCP's Retransmission Timer*. RFC2988, 2000.
[16] Proakis, J.G., *Digital communications(Fourth Edition)*. 2001: McGraw-Hill Inc.
[17] Raheli, R., A. Polydoros, and C.-K. Tzou, *Per-Survivor Processing: A General Approach to MLSE in Uncertain Environments.* IEEE Trans. ON COMMUNICATIONS, 1995. **43**(2/3/4): p. 354-364.
[18] Seshadri, N., *Joint Data and Channel estimation Using Blind Trellis Search Techniques.* IEEE Trans. ON COMMUNICATIONS, 1994. **42**(2/3/4): p. 1000.
[19] Zhang, Y., et al. *On the Characteristics and Origins of Internet Flow Rates*. in *SIGCOMM '02*. 2002.