Passive Overall Packet Loss Estimation at the Border of an ISP

Haoliang Lan[1], Wei Ding[1] and YuMei Zhang[2]

[1]School of Computer Science and Engineering, Southeast University

Nanjing, 211189 – CHINA

[email: hllan@njnet.edu.cn]

[2]School of Big Data and Information Engineering, Guizhou University

Guiyang, 550025 – CHINA

[email: yumeiz@yahoo.com]

*Corresponding author: Haoliang Lan

Received October 20, 2017; revised March 5, 2018; accepted March 14, 2018; published July 31, 2018

Abstract

In this paper, a heuristic method that leverages packet traces captured at the entire boarder of an ISP to distinguish and estimate the overall packet loss within an ISP's management domain (*Intra_Path_Loss*) and that in the outside Internet (*Inter_Path_Loss*) is proposed. Our method is inspired by that packet losses happened at different locations will cause different TCP sequence number patterns at the border of an ISP. Thereby, we leverage these TCP sequence number patterns to build a series of heuristic rules to estimate *Intra_Path_Loss* and *Inter_Path_Loss*, respectively. We do this work with an eye towards showing that the overall packet losses defined and estimated in this paper can provide the operators with some valuable information to help them precisely grasp the overall performance of network paths and narrow down the range of network anomalies. The proposed method is rigorously validated with simulations, and finally the results from a regional academic network JSERNET[1] verify its effectiveness and practicability.

[1] JSERNET (Jiangsu Education and Research Network) is a regional academic network of CERNET. It covers more than 100 research units and universities, and its backbone bandwidth increased from OC-48 to OC-192 in January 2006.

http://doi.org/10.3837/tiis.2018.07.010 **ISSN : 1976-7277**

Keywords: Pakcet Loss rate, Estimation, Network performance management, Network measurement

1. Introduction

As connection-oriented and reliable transport protocol, TCP has a natural response to changes in network performance. Therefore, evaluating network performance with TCP has always been a research hotspot within the academia. Meanwhile, the packet loss rate as a key performance metric is important for both the operators and the end users. As operators, accurate measurement for packet loss is crucial for classic network management tasks, such as traffic engineering and capacity planning, while as end users, the estimation for packet loss also enables them to achieve the monitoring for both quality of service (QoS) and quality of experience (QoE) [1][2][3]. All along, since the doorsill for obtaining the research data is lower, a lot of research has mainly focused on the techniques of end-system packet loss estimation. For instance, Madhyastha *et al*. [4] predict packet loss between arbitrary Internet hosts by composing the performance of the measured segments. Friedl *et al*. [5] estimate packet losses with 100% accuracy by measuring data segments at two endpoints of a connection. Basso *et al*. [6] derive a model Inv-M from the well-known Mathis equation to estimate application-level packet loss, and later they [7] further improve the accuracy of Inv-M by proposing a new model L-Rex. Silveira *et al*. [8] estimate loss rates and their confidence intervals by building a Hidden Semi-Markov Model (HSMM) for the measurement process. Hu *et al*. [9] present a new packet loss estimation technique by making use of the user_data field of video. Compared with the rich papers in this area, we only list a small part of recent research. Such methods have a common feature, viz., they are all based on the information available from the TCP sender-side and/or receiver-side, which allows the users to measure networks in which they only control the endpoints of a TCP connection. However, due to the traffic shaping and the events violating network neutrality, different TCP connections may experience vastly different loss rates. From the perspective of network performance management, the packet loss rate obtained by such methods is not suitable for evaluating the overall packet loss status of the monitored network.

This paper adds to the body of estimation techniques by detailing and validating a method, which can be used by ISPs to distinguish and evaluate *Intra_Path_Loss* and *Inter_Path_Loss*. Compared with previous work, our method has several attractive properties, including:

- It doesn't have the issues of difficult deployment and collaborative operation consistency that exist in the methods that need to collect data from different vantage points [4][5].
- The network sometimes processes each TCP stream unfairly, i.e., violating the principle of network neutrality [6]. The packet loss rate experienced by a single TCP connection may not be able to reflect the real packet loss status of the measured network. Compared with traditional end-to-end measurements [4][5][6][7][8][9][10], it can estimate packet loss rates of both the individual end-to-end connection and the aggregated traffic.
- By dividing the destination addresses, it can obtain the overall loss rate between the measured network and some specific network. Meanwhile, it can also distinguish the packet loss within an ISP's management domain and that in the outside Internet. All these provide the network operators with the possibility for conducting fine-grained analysis to the packet loss to precisely grasp the performance of the network path.

First, to evaluate and distinguish the overall packet losses within an ISP and that in the outside Internet, *Intra_Path_Loss* and *Inter_Path_Loss* are defined in this paper. As we will show later, they can provide the network operators with some valuable information about whether the network performance is maintained at a normal level or there are problems within an ISP's management domain and/or in the outside Internet. In addition, for some abnormal events (such as earthquake, etc.), we can also study their impact by conducting fine-grained analysis to the corresponding *Intra_Path_Loss* or *Inter_Path_Loss*.

Second, heuristic techniques that estimate *Intra_Path_Loss* and *Inter_Path_Loss* are introduced and validated, they are inspired by that packet losses happened at different locations (within an ISP or in the outside Internet) will cause different TCP sequence number patterns at the border of an ISP. Actually, these different TCP sequence number patterns are related with different state transitions of the state machines in the sender-side and receiver-sider that caused by packet losses at different locations. Therefore, our goal is to build a series of heuristic rules to reflect these different TCP state transitions to further estimate packet losses happened at different locations. Due to heuristic rules

always lack a general proof, thus we validate them using various simulations with practically relevant parameters.

Finally, we use the proposed method to analyze the long-term packet loss status of a regional academic network JSERNET between 2005 and 2016 with an eye towards for verifying its practicability and effectiveness.

The rest of the paper is organized as follows: *Intra_Path_Loss* and *Inter_Path_Loss* are defined in Section 2. Then Section 3 details and validates the corresponding packet loss estimation algorithms. In Section 4, the results from a regional academic network are presented and analyzed. Finally, Section 5 concludes the paper.

2. Definition of the Overall Packet Loss Rates

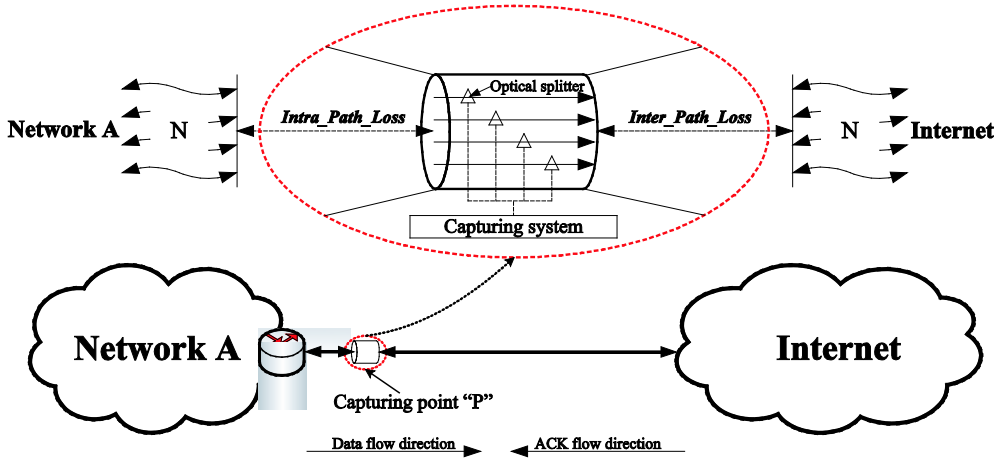

Fig. 1. Model for evaluating overall packet losses

Fig. 1 shows the simple model for evaluating *Intra_Path_Loss* and *Inter_Path_Loss*. Each network path that crosses through the edge of network A is divided into two segments by the capture point "P". The N network path segments before "P" can be used to evaluate the overall packet loss within the management domain of A, while the N network path segments after "P" can be used to evaluate the overall packet loss in the outside Internet. The concept of "before" and "after" in actual network references data flow direction. Accordingly, *Intra_Path_Loss* is defined as the weighted sum of the N network path segment loss rates before "P":

$$Intra\_Path\_Loss = \sum_{i=1}^{N} P_{before\_i} * W_i \tag{1}$$

where P_{before_i} is the loss rate of the i[th] network path segment before "P", and W_i is the weighted value of the i[th] network path. Here, W_i is calculated as:

$$W_i = \frac{N_{total\_i}}{\sum_{i=1}^{N} N_{total\_i}} \tag{2}$$

where N_{total_i} is the number of data packets belonging to the i[th] network path.
Similarly, *Inter_Path_Loss* is defined as:

$$Inter\_Path\_Loss = \sum_{i=1}^{N} P_{after\_i} * W_i \tag{3}$$

where P_{after_i} is the loss rate of the i[th] network path segment after "P".

From discussion above, we can see that the key to obtain *Intra_Path_Loss* and *Inter_Path_Loss* lies in estimating P_{before_i} and P_{after_i}. Thus next our goal is to estimate the loss rates of the two path segments separated by "P" and to be as accurate as possible.

3. Methodology and Validation

See **Fig. 1** and consider the location that a data packet is lost in a TCP connection, it may be lost before or after "P". A data packet is lost at different locations will cause different TCP sequence number patterns at "P", which can be leveraged to estimate packet losses on the network paths before and after "P".

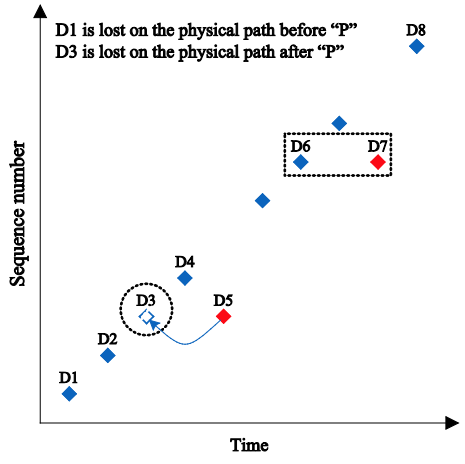

Fig. 2. Packet loss detection principle

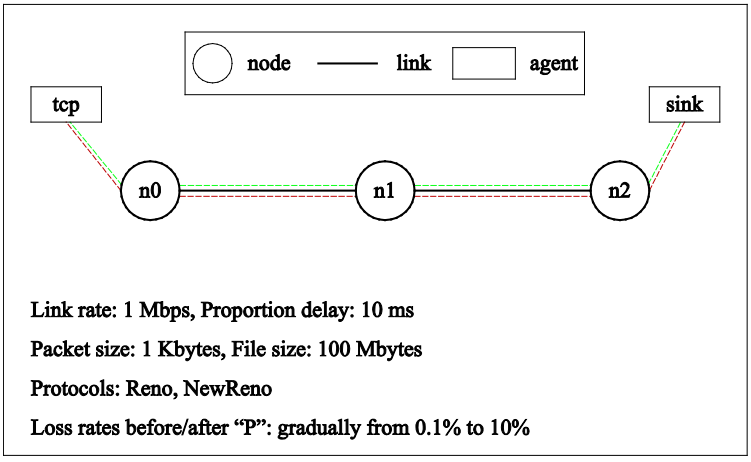

Fig. 3. Simulation scenario

3.1 Packet Loss Estimation for Network Path before "P"

In this section, the *A*lgorithm for *E*stimating *P*acket *L*osses on *N*etwork *P*ath before "P" ($AEPLNP_{before}$) is introduced. $AEPLNP_{before}$ estimates packet losses by building a series of heuristic rules that aim to accurately reflect the state transitions of TCP congestion state machine in the sender-side caused by packet losses on network path before "P". Consider the situation that the server sends a row of data segments within the send window (e.g., D_1, D_2, D_3 and D_4 in **Fig. 2**) and a data segment (e.g., D_3 in **Fig. 2**) is lost on the network path before "P", then the lost segment will cause a "hole" in the TCP data sequence. When the TCP sender retransmits the lost segment (e.g., D_5 in **Fig. 2**) to repair this packet loss, the out-of-order segment D_5 will appear at "P" and falls into the hole caused by the lost segment D_3. Therefore, $AEPLNP_{before}$ can detect a packet loss happened before "P" by observing an out-of-order segment falls into a hole.

To implement $AEPLNP_{before}$, the data sequence of a TCP connection is described with set $S_{original}=\{<S_1, L_1, I_1>, \ldots, <S_n, L_n, I_n>\}$, where $\forall\, i \in N$, $<S_i, L_i, I_i>$ denotes the i^{th} data segment appearing at "P", S_i denotes the sequence number of $<S_i, L_i, I_i>$, L_i denotes the byte length of $<S_i, L_i, I_i>$, I_i denotes the identification in the IP header of $<S_i, L_i, I_i>$. For each $<S_i, L_i, I_i>$, where $i > 2$, we select elements that are before $<S_i, L_i, I_i>$ in $S_{original}$ to construct set $S_{check}=\{<S_1, L_1, I_1>, \ldots, <S_m, L_m, I_m>\}$, where $1 \le m < i$, $\forall\, j \in N$, $S_j \le S_{j+1}$. Then according to the description in **Fig. 2**, the number of packet losses on the network path before "P" in a TCP connection can be calculated as:

$$N_{before} = \sum_{i=3}^{n} N_i \tag{4}$$

where N_i is determined as following:

$$N_i = \begin{cases} 1: \ \exists <S_j, L_j, I_j> \in S_{check}, S_{j-1}+L_{j-1}<S_j \wedge S_{j-1}<S_i<S_j \\ 0: \text{otherwise} \end{cases} \tag{5}$$

In order to evaluate the simple algorithm, we implemented and validated it with packet traces obtained from simulations. The simulation was carried out using Network Simulator (NS-2) [11]. The simulated scenario, shown in **Fig.3**, consists of three nodes: n0 (server), n1 (capture point) and n2 (client). The link rate and proportion delay was set to 1Mbps and 10 ms, and the packet size was set to 1 Kbytes. TCP Reno and NewReno were simulated separately, and total 20 TCP transfers were scheduled between the server and the client to transfer a fixed-size file (100 Mbytes) during the simulation. For each transfer, we collected packet traces from the three points and compared them to obtain the accurate loss rates before and after "P". On the other hand, we run our algorithms on the packet traces collected from the capture point to get the estimated loss rates. Eventually, the relative error is used to evaluate the accuracy of the simple algorithm, which is calculated as the absolute difference between the estimated loss rate and the accurate loss rate divided by the accurate loss rate:

$$Error_{relative} = \frac{|Loss_{estimated} - Loss_{accurate}|}{Loss_{accurate}} \tag{6}$$

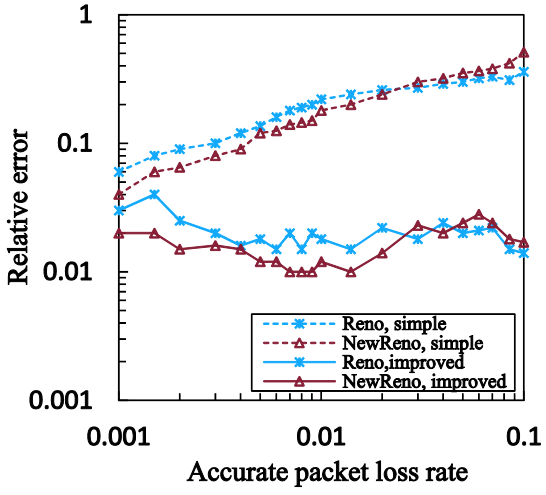

Fig. 4. Simulation results of $AEPLNP_{before}$

The simulation result is shown in **Fig. 4**. As can be seen, in the majority of parameter space, $AEPLNP_{before}$ has higher accuracy when the loss rates are not high. For high loss rates, the worst-case relative error was 0.51 and the corresponding accurate loss rate was 0.1, which implies the absolute error was 5.1%. Indeed, the error was mainly caused by that the simple rule is not able to cover all sequence number patterns caused by packet losses on the network path before "P". When the loss rate becomes higher, more samples not covered by the simple rule appeared, which made the estimation error become higher. In most cases, it is hard and even impossible to take all possible sequence number patterns into consideration just relying on the packet traces obtained from the capture point. But fortunately, what we are interested in is not the exact number of packet losses but how to make the estimated loss rate more accurate. Therefore, what we should do is identifying and eliminating the main error sources. For the estimation errors of the simple algorithm, we think the following factors associated with packet dynamics [12] can explain to some extent.

- **Packet reordering:** Here packet reordering refers to a phenomenon that an earlier sent packet arrives at "P" later than one or more later sent packet(s). In this case, the segment arriving later is out-of-order and falls into a hole caused by the segments arrived earlier, which fools our algorithm.
- **Repeated packet losses:** If a packet is lost multiple times on the network path before "P", our algorithm can only detect up to one time, because in this case, it can only detect one data segment falling into the corresponding hole.

Except the error sources listed above, there must be others. For instance, the entire window segments are lost will not cause any visible hole in the TCP data sequence, and it typically happens at time when the sending window size is not big. However, as we will see, the above two are the major ones. In order to exclude the effect of packet reordering, we make use of the information contained in the identification in the IP header (hereinafter referred to as *IP-Id*). To our knowledge, most operating systems increase the value of *IP-Id* by one after completing every packet sending. So if there is a *decrease* of the *IP-Id* (e.g., in **Fig. 2**, if the *IP-Id* of D_5 is less than that of D_4), we think it is caused by packet reordering. In addition, consider the range identified by the *IP-Id* is 0 to 65535, when the number of the data segments in a TCP connection is more than 65536, the *IP-Id* will appear cyclic reuse, which can also cause a decreased *IP-Id*. Therefore, the maximum decreased value is set to handle the *IP-Id* cyclic reuse. That is if the decreased value exceeds a pre-defined threshold (e.g., 50000), the algorithm assumes the *IP-Id*

cyclic reuse has occurred.

To cope with repeated packet losses, we search for packet losses that *loss periods* are 1. The loss period, defined in RFC3357, refers to the number of packet losses that occur in a row. Analyze the packet traces collected by the Flow_Mirror[2], we found that roughly 60% loss periods in a TCP connection are 1, the repeated packet losses occupied about 5% and the number of consecutive packet losses is typically less than 3. Therefore, if M represents the number of packet losses that the loss periods are 1, then the algorithm assumes the number of repeated packet losses in a TCP connection is:

$$N_{repeated} = \left\lfloor M * \frac{5\%}{60\% + 5\%} \right\rfloor * (3 - 1) \tag{7}$$

After excluding the effect of the two factors above, the equation (4) is corrected as:

$$N_{before} = \sum_{i=3}^{n} N_i + N_{repeated} \tag{8}$$

where, N_i is corrected as:

$$N_i = \begin{cases} 1: \ \exists <S_j, L_j, I_j> \in S_{check}, S_{j-1}+L_{j-1}<S_j \wedge S_{j-1}<S_i<S_j \wedge \left(I_{j-1}<I_i \vee I_{j-1}-I_i \geq 5000\right) \\ 0: \text{otherwise} \end{cases} \tag{9}$$

In order to validate the improved algorithm, a simulation with the same settings was performed. The results is also shown in **Fig. 4**. As can be seen, the relative errors of the improved algorithm are controlled within 4% in most cases. Meanwhile, it increases the estimation accuracy for both low-loss and high-loss situations. Moreover, the improved algorithm has a more stable estimation performance with the increase of the loss rate. All these improvements can be mainly attributed to that the rules dealing with packet reordering and repeated packet losses reduced the false positives and false negatives, respectively. Eventually, the improved $AEPLNP_{before}$ is given in Algorithm 1.

[2] The Flow_Mirror is a system that we developed based on TCPDUMP, its client sides periodically perform TCP transfers with instances of itself. During the transfers, the system collects traces at two endpoints of a TCP connection and store them on the fixed server. Some of the collected data is named "IPTAS TCP Base Database" and published on: http://iptas.edu.cn/src/system.php, so anyone is free to reuse them for research purposes.

Algorithm 1: $AEPLNP_{before}$

```
//Preprocessing stage
Loss_before = Loss_period = 0
for pkt in trace of a TCP connection
    if pkt.is_Data()
       pkt.addArrary(S)
    else
       continue
end for
for elmt in S
    if elmt.fall_Hole()
      if elmt.reorder()
           continue
      else
           Loss_before += 1
           Loss_period = period(elmt, S)
      end if
    else
           continue
    end if
end for
Loss_repeated = ⌊Loss_period/13⌋ * 2
return Loss_before
```

3.2 Packet Loss Estimation for Network Path after "P"

In this section, the *A*lgorithm for *E*stimating *P*acket *L*osses on *N*etwork *P*ath after "P" ($AEPLNP_{after}$) is introduced. Similarly, $AEPLNP_{after}$ estimates packet losses also by building a series of heuristic rules that aim to accurately reflect the state transitions of the state machines in the sender-side and receiver-side caused by packet losses after "P".

Given that a packet is lost after "P" and is retransmitted, in this case, both the original and retransmitted packets should appear at "P". For instance, D_6 in **Fig. 2** is lost on the network path after "P", D_7 is its retransmission and appears at "P". Therefore, it is a natural idea that leverage retransmissions at "P" to estimate packet losses on the network path after "P". But unfortunately, the situation is a little more complicated than what we have described in **Fig. 2**. Since the flaw of TCP retransmission mechanisms can cause spurious retransmissions in many cases[13][14], a simple statistics of retransmissions at "P" will yield an overestimate for packet loss after "P". Thus, to accurately estimate packet losses, we need to develop rules to detect and exclude the spurious retransmissions.

Although some methods detecting spurious retransmissions were proposed [13][14][15][16], but they are not suitable for use at point "P". For instance, the Eifel algorithm detects the spurious retransmissions leveraging TCP timestamp, but not all connections enable the TCP timestamp option in actual network. Additionally, obtain exact timestamp in the middle of the network (e.g., at boarder of an ISP) is difficult. Again, the F-RTO algorithm needs the knowledge about the data segments that have been sent within the same window, while it is not available when only using packet traces captured at "P". Hence, how to detect the spurious retransmissions at "P" becomes key to our algorithm. If the number of spurious retransmissions in a TCP connection is determined, then the number of packet losses on network path after "P" can be simply calculated as the number of retransmitted segments minus the number of spurious retransmissions:

$$N_{after} = N_{retransmitted} - N_{spurious} \tag{10}$$

Next, like the literatures [15] and [16], we try to leverage information contained in the ACK stream to identify the spurious retransmission, but the specific approach is different.

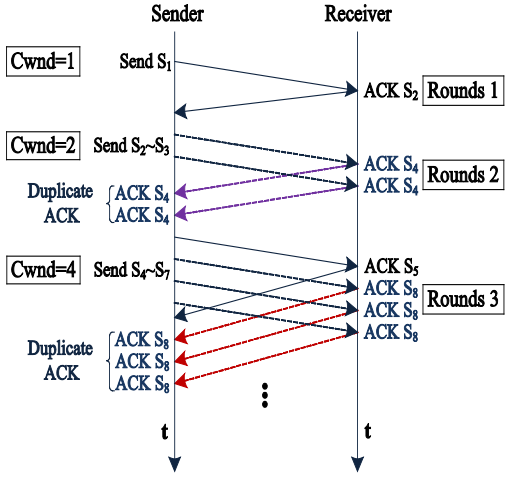

Fig. 5. Retransmission pattern for SCA after RTO

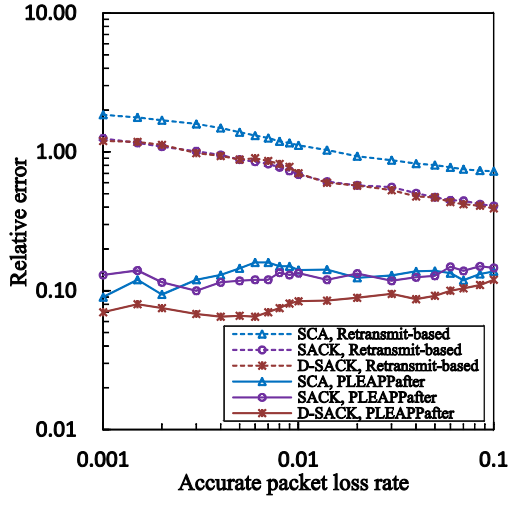

Fig. 6. Simulation results of $PLEAPP_{after}$

In fact, for different TCP acknowledgement mechanism, the form of ACKs is different. Accordingly, different rules will be used to recognize the spurious retransmissions in this paper. Currently there are three kinds of TCP acknowledgement mechanisms:

- **Standard Cumulative Acknowledgment (*SCA*):** According to RFC 5681, it cooperates with the basic TCP congestion control mechanisms to repair the packet losses (e.g., Reno, NewReno, BIC and CUBIC, etc.).

- **Selective Acknowledgment (*SACK*):** It uses the SACK blocks to acknowledge out-of-order segments that have arrived at the receiver and not covered by the acknowledgement number. The SACK combines with a selective retransmission policy at the sender-side to repair the packet losses and reduce the spurious retransmissions. More details about SACK are available from RFC2018.
- **Duplicate Selective Acknowledgment (*D-SACK*):** This version, described in RFC2883, is an extension to the SACK. It allows the receiver to inform the sender about segments that have already arrived more than once. Therefore, we can accurately determine a spurious retransmission with an ACK containing D-SACK information.

For *SCA*, given that total seven segments (S_1, S_2, S_3, S_4, S_5, S_6,and S_7) are sent, and only S_3, S_6 and S_7 arrive at the receiver. **Fig. 5** shows the retransmission process after RTO expiration in this case:

a) The sender retransmits S_1, and the receiver sends an ACK for expecting S_2 after receiving S_1.
b) Upon receiving the ACK for expecting S_2, the cwnd (congestion window) at the sender-side increases to two. Then, the sender retransmits S_2 and S_3 even through it has no any knowledge about whether S_3 is lost or not.
c) After receiving S_2, the receiver sends the ACK for expecting S_4 since S_3 has already arrived at the receiver. When the receiver receives the spurious retransmission of S_3, the *duplicate* ACK for expecting S_4 is generated and transmitted.
d) Similarly, when S_5, S_6,and S_7 arrive at the receiver, they will also cause duplicate ACKs for expecting S_8.

Inspired by this, the basic principle of our detection rule for *SCA* is that if the retransmission is spurious, the acknowledgements for the original transmission and the spurious retransmissions should all appear at "P".

The ACK sequence of a TCP connection is described with set $A_{sca}=\{<A_1, B_1>, \ldots, <A_n, B_n>\}$, where $\forall\, i \in N$, $<A_i, B_i>$ denotes the ith ACK appearing at the capture point, A_i denotes the acknowledgement number of $<A_i, B_i>$. Then N_{after} can be calculated as:

$$N_{after} = N_{retransmitted} - \sum_{i=1}^{n} B_i \tag{11}$$

where B_i denotes the following:

$$B_i = \begin{cases} 1: \exists A_j \in A_{sca}, j<i \wedge A_j=A_i \\ 0: \text{otherwise} \end{cases} \tag{12}$$

On the other hand, for the duplicate ACK of SCA, we can prove the following:

Proposition 1. The duplicate ACK is caused by either spurious retransmission or packet reordering.

Proof. In order to let the sender-side sent the data segment D that is suspected to be lost as soon as possible and avoid RTO expiration, the fast retransmission requires the receiver to immediately generate an ACK for expecting D upon receiving an out-of-order data segment. Therefore, if the received data is out-of-order, then a duplicate ACK will be generated. In contrary, if the received data is orderly and non-retransimitted, then it will cause a new ACK. Else if the received data is orderly and necessary retransmission, it will cause a new ACK, and if it is orderly and spurious retransmission, the duplicate acknowledgement will be generated.

Therefore, for SCA, duplicate ACKs can be used to detect spurious retransmissions, but need to exclude the effect of packet reordering, i.e., duplicate ACKs caused by packet reordering shouldn't be taken into account. As we know, the essence that packet reordering can generate duplicate ACK is fast retransmission mechanism. Again, sufficient packet reordering will cause fast retransmission (typically over a three duplicate ACKs). Therefore, for a duplicate ACK, if the number of times it appears at the capture point is greater than or equal to 4, we assume it is caused by packet reordering. Thus we further denote $A_{sca}=\{<A_1, B_1, N_1>, \ldots, <A_n, B_n, N_n>\}$, where $\forall\ i \in N$, $<A_i, B_i, N_i>$ denotes the ith ACK appearing at the capture point and N_i denotes the number of times $<A_i, B_i, N_i>$ appears at the capture point. To exclude the effect of packet reordering, B_i is corrected as:

$$B_i = \begin{cases} 1: \exists A_j \in A_{sca}, j<i \wedge A_j=A_i \wedge N_i \leq 3 \\ 0: \text{otherwise} \end{cases} \tag{13}$$

For SACK, with the help of SACK blocks, the TCP sender may not conduct spurious retransmission in the case like **Fig. 5**. However, the spurious retransmissions still exist for SACK. For example, the sender always clear the scoreboard of SACK blocks after RTO expiration, while the receiver is not able to refill the scoreboard of the sender-side since it always only acknowledges the most recently transmitted segments. In this case, the sender has no any knowledge about the out-of-order segments that have already reached the receiver, which results in inevitable spurious retransmissions. To detect the spurious retransmissions for SACK, we reversely analyze the effect of spurious retransmission on the receiver, viz., if the retransmission is spurious, the buffer state of the receiver-side would not be changed. For SACK, the ACK stream of a TCP connection is described with set $A_{sack}=\{<A_1, S_1, I_1>, \ldots, <A_n, S_n, I_n>\}$, where $\forall\ i \in N$, $<A_i, S_i, I_i>$ denotes the ith

ACK appearing at "P", A_i denotes the acknowledgement number of $<A_i, S_i, I_i>$, S_i denotes the set of SACK blocks in the i^{th} ACK, and I_i denotes the identification in IP header of the i^{th} ACK. Regarding the spurious retransmission for SACK, we can prove the followings:

Proposition 2. $\exists <A_i, S_i, I_i>$ caused by either spurious retransmission or ACK reordering, $A_i \leq \text{MAX}(A_1, A_2, \ldots, A_{i-1}) \wedge S_i \subset \cup_{j<i} S_j$

Proof. If the received data is non-retransmitted, then due to the SACK field, no matter the received data is orderly or not, a new ACK will be generated. If the received data is retransmitted and spurious, then it would not change the buffer state of the receiver-side. Thus, the information contained in the ACK caused by this spurious retransmission will become redundant compared with that contained in the ACKs having already left the receiver. When this ACK arrives at "P" and compared with the ACKs that have already reached "P", the redundant information makes it does not advance in acknowledgement number and does not contain new SACK blocks. Else if the the received data is retransmitted and non-spurious, then a new ACK will be generated as the buffer state of the receiver-side is changed. On the other hand, consider the generated ACK. If it is orderly, then the order in which it arrives at the capture point will be consistent with the order in which it was sent at the receiver-side. Else if the ACK is out-of-order when arriving at "P" (different from packet reordering, here called ACK reordering), in this case, when compared with the information contained in the ACKs that were sent later but have already arrived at "P" earlier, the information contained in this ACK may become redundant or even less.

As discussed above, spurious retransmission can produce $<A_i, S_i, I_i>$ with the characteristic of $A_i \leq \text{MAX}(A_1, A_2, \ldots, A_{i-1}) \wedge S_i \subset \cup_{j<i} S_j$. On the other hand, we can conclude that not all ACKs with this characteristic are caused by spurious retransmissions. Therefore, we can leverage this characteristic to detect spurious retransmission for SACK but need to exclude the effect of ACK reordering. Faced with this, just like what we have done in Section 3.1, the *IP-Id* is used to exclude the effect of ACK reordering. Concretely, we have the following steps:

Step 1: for $\forall <A_i, S_i, I_i>$, where $A_i \leq \text{MAX}(A_1, A_2, \ldots, A_{i-1}) \wedge S_i \subset \cup_{j<i} S_j$, we first make $M_i = \text{MAX}(A_1, A_2, \ldots, A_{i-1})$;

Step 2: according the obtained M_i, we search the first i-1 elements in A_{sack} to find $<A_j, S_j, I_j>$ where $A_j = M_i$;

Step 3: we determine that $<A_i, S_i, I_i>$ is caused by spurious retransmission when and only when $I_i > I_j \vee I_j - I_i \geq 5000$.

The solution described above doesn't necessarily exactly exclude the effect of ACK reordering, but it is expected to limit the error.

Finally, if the receiver supports D-SACK, it will send an ACK containing DSACK blocks for each duplicate data segment. Therefore, $AEPLNP_{after}$ can use the ACK containing DSACK information to accurately determine a spurious retransmission. The specific rules recognizing DSACK blocks can reference to RFC2883.

Likewise, we implemented the algorithm $AEPLNP_{after}$ and validated it with packet traces obtained from simulations. The simulations had the same settings as shown in section 3.1, meanwhile we simulated three different types of TCP acknowledgement mechanisms, respectively. The results are shown in **Fig. 6**, for comparison, the results of retransmit-based estimation were also plotted.

As can be seen, the performance of retransmit-based estimations were worst, which revealed the flaw of current TCP retransmission mechanisms and the necessity for excluding the spurious retransmissions when estimating packet losses on the network path after "P". Additionally, we can also see that the retransmit-based estimation for SACK and D-SACK was better than that for *SCA*. This can be attributed to the SACK field that reduced the spurious retransmissions at the sender-side to some extent. For our algorithm, compared with the retransmit-based estimation, it observably reduced the estimation errors on the whole. Specifically, it achieved optimum performance on D-SACK transfers with relative error of less than 10%. Analyze the errors for D-SACK, we found they are all belonged to overestimates. This reflected the fact that for D-SACK, the segment determined as spurious retransmission does reach the receiver more than once. Therefore, it can be inferred that the estimation errors for D-SACK were all caused by the loss of ACKs that specify spurious retransmissions. For SCA and SACK, the relative errors were also controlled around 10% after excluding the spurious retransmissions. Different from D-SACK, for SCA, our algorithm mainly excluded the spurious retransmissions caused by slow start strategy after RTO expiration, and the result shows it achieved a good effect. Thus, we can conclude that for SCA, most of spurious retransmissions are mainly caused by the flaw in slow start strategy after RTO expiration. While for SACK, by leveraging the ACKs having characteristic described in step 1, our algorithm also locates the majority of spurious retransmissions caused by sender-side's scoreboard information missing. Thus given only the information available from the capture point, $AEPLNP_{after}$ has achieved a good estimation for packet losses happened after the capture point. While for the estimation errors of SCA and SACK, we think the following factors can explain in a certain extent.

- **Spurious fast retransmissions:** $AEPLNP_{after}$ can't deal with the spurious fast retransmission caused by sufficient packet reordering. It's included in N_i and makes B_i zero at the same time, which skews our estimation results.
- **Lost ACKs:** An ACK is lost in network may cause the corresponding spurious retransmission cannot be recognized.
- **Packet/ACK reordering:** As we have mentioned, packet/ACK reordering can lead to an ACK seems to be caused by spurious retransmission. Although the rule for SCA and the *IP-Id* for SACK were used to exclude its negative effect, but they can only limite the error and cannot make our estimate exactly right.

Eventually, the algorithm $AEPLNP_{after}$ is given in Algorithm 2.

Algorithm 2: $AEPLNP_{after}$

```
//Preprocessing stage
Loss_after = Retr = R_ack = R_sack = R_dsack = 0
Sup_sack = Sup _dsack = false
for pkt in trace s of a TCP connection
    if pkt.is_Data() and pkt.rep_Check(pkt.seq_No())
                    Retr += 1
                    continue
    else if pkt.is_Ack()
            if not Sup_sack
              if ack_Redundant(pkt) then
                    R_ack += 1
                    Loss_after = Retr - R_ack
              else
                    continue
              end if
            else if Sup_dsack and dsack_Redundant(pkt)
                    R_dsack += 1
                    Loss_after = Retr - R_dsack
            else if not Sup_dsack and sack_Redundant(pkt)
                    R_sack += 1
                    Loss_after = Retr - R_sack
            else
                    continue
            end if
    else
                    continue
    end if
end for
return Loss_after
```

Note: According to our research [17], although the entire border traffic of the monitored network JSERNET is captured, but among the network environment of multiple-operators, due to the network management capacity of some small access units is limited, the misconfiguration will cause asymmetric routing at the border of JSERNET and eventually result in small amount of artificial one-way traffic at the edge of the monitored network. For one-way traffic, its causes may be benign (unreachable services, misconfiguration, etc.) or malicious (attacks). Among them, the malicious is the major and doesn't need to be considered when estimating packet losses. For the benign, the ACKs cannot be leveraged, so we use the retransmissions at point "P" to estimate packet losses happened after "P".

4. Results From JSERNET

In this section, we apply the proposed method on a massive traffic traces captured from the border of a regional academic network JSERNET to analyze its long-term packet losses. The traffic traces were captured by the tool WATCHER which we developed and maintain, and it was designed to capture all the traffic that crosses the border of JSERNET destined to or coming from the Internet. The statistical information of traffic traces used for packet loss analysis is shown in **Table 1**.

Table 1. Traffic traces used for packet loss analysis

Time	2005-11-10	2006-12-13	2007-1-6	2008-12-14	2009-11-14	2010-11-14	2011-3-11	2012-12-22	2013-02-22	2014-05-09	2015-11-10	2016-10-18
Total gigabytes	845.51	903.52	746.48	862	959	942	1139.44	1021	806.79	2120.93	3483.23	5191.68
Total Packets (1e9)	12.34	11.66	13.35	10.97	15.12	14.85	17.99	16.11	12.74	33.49	11.01	16.41

4.1 Data Sanitization

For two-way traffic, packets containing source IPs that are not those really assigned to their sending host, i.e., *spoofed traffic* [18], should not be taken into account when estimating packet losses. Therefore, we use the methodology in the literature [18] to exclude these traffic. It has two steps: firstly, find out the two-way TCP connections; secondly, remove connections with too few packets (5 packets) or bytes (80 bytes). For one-way traffic, we use the classifier described in the literature [19] to pick out the traffic caused by asymmetric routing.

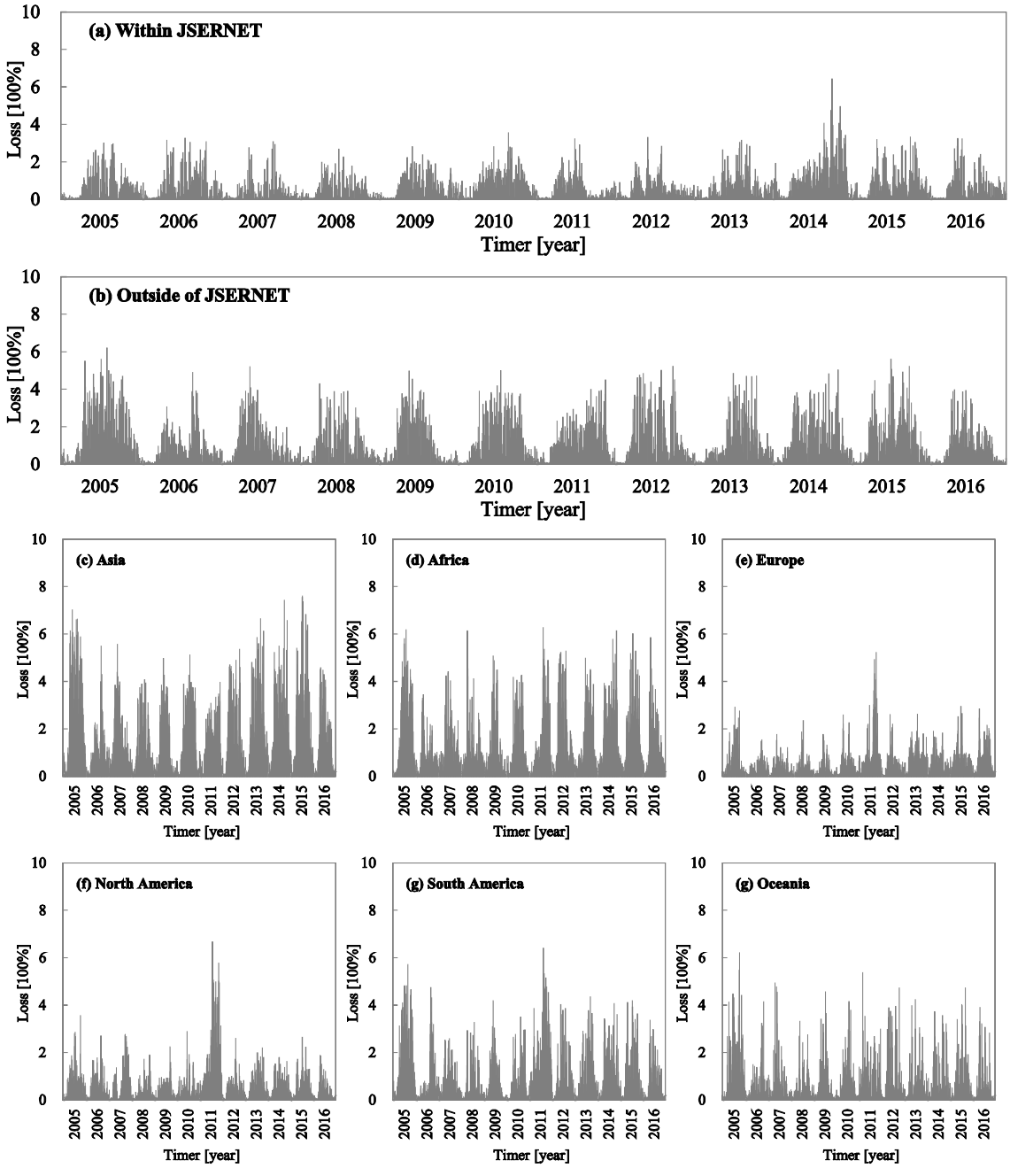

Fig. 7. Packet loss status of JSERNET

4.2 Results and Analysis

The results are shown in **Fig. 7** (each point represents 10-minute interval). Concretely, in **Fig. 7 (a)**, the downlink overall loss rate after "P" is shown. As can be seen from **Fig. 7 (a)**, the overall loss rate within JSERNET was kept in a normal level on the whole with small fluctuations except that of 2014. Analyze the abnormal loss rates in 2014, we found that when the abnormal loss rates appeared, the reports for DDOS attacks [20] was generated almost simultaneously. Therefore, this indicates that the network congestion caused by DDOS attacks may have some relationship with the abnormal loss rates. In addition, the downlink overall loss rate before "P" is shown in **Fig. 7 (b)**. The result in **Fig. 7 (b)** is intuitive that the overall loss rate in the outside of JSERNET in 2007 had a visible decline compared with that of 2006. The decline can be attributed to the upgrade of JSERNET's backbone bandwidth in January 2006 (from OC-48 to OC-192), and the declined packet loss rates indicate that this upgrade made the original congestion line become smooth.

Finally, **Fig. 7 (c) ~ (h)** shows the overall loss rates from Asia and other continents to JSERNET. In **Fig. 7 (c) ~ (h)**, we can see that except 2011, the overall loss rates from Europe and North America to JSERNET were lower than other continents and even lower than Asia. This, we believe, reflected the fact that Europe and North America are the current internet hubs. Comparing with other continents, the good network infrastructure and perfect connectivity within these two regions produced the lower loss rates. On the other hand, **Fig. 7 (b)** shows the overall loss rate in the outside internet in 2011 was remained at a normal level on the whole, but the overall loss rates for North America, South America and Europe in 2011 increased instead. This is due to that the data for 2011 were captured during the earthquake happened in eastern seas of Japan, and these data were specifically picked out to study the impact of the earthquake. As we all know, this earthquake caused huge damage to the undersea cable, which resulted in large-scale routing revocation and routing table reconstruction. See **Fig. 7 (e) ~ (g)**, from the increased loss rates in 2011, we can infer that this earthquake mainly led to the interruption and congestion of some connections between JSERNET and North America, South America and Europe.

5. Conclution

In order to distinguish and evaluate the overall packet loss within an ISP's management domain and that in the outside Internet, *Intra_Path_Loss* and *Inter_Path_Loss* are defined in this paper. Correspondingly, their estimation algorithms were presented and rigorously validated with simulations. Finally, the results from a regional academic network JSERNET demonstrated that the overall packet losses defined and estimated in this paper can provide the operators with some valuable information to help them precisely grasp the overall performance of network paths and narrow down the range of network anomalies.

Acknowledgements

This paper was sponsored by the National Grand Fundamental Research 973 program of China (2009CB320505); the National Nature Science Foundation of China (60973123).

References

[1] Feng, B., Zhou, H., Zhang, M., and Zhang, H, "Cache-Filter: A Cache Permission Policy for Information-Centric Networking," *KSII Transactions on Internet and Information Systems* (*TIIS*), vol.9, no.12, pp. 4912-4933, December, 2015. Article (CrossRef Link).

[2] Zhou Liang, "On data-driven delay estimation for media cloud," *IEEE Transactions on Multimedia*, vol.18, no.5, pp. 905-915, May, 2016. Article (CrossRefLink).

[3] Zhou, Liang, "QoE-driven delay announcement for cloud mobile media," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.27, no.1, pp. 84-94, January, 2017. Article (CrossRefLink).

[4] Madhyastha, H. V., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., & Venkataramani, A, "iPlane: An information plane for distributed services," in *proc. of the 7th symposium on Operating systems design and implementation*, pp. 367-380, November, 2006. Article (CrossRefLink).

[5] Friedl, A., Ubik, S., Kapravelos, A., Polychronakis, M., & Markatos, E. P, "Realistic passive packet loss measurement for high-speed networks," in *proc. of International Workshop on Traffic Monitoring and Analysis*, pp. 1-7, May, 2009. Article (CrossRefLink).

[6] Basso, S., Meo, M., Servetti, A., & De Martin, J. C, "Estimating packet loss rate in the access through application-level measurements," in *proc. of the 2012 ACM SIGCOMM workshop on Measurements up the stack*, pp. 7-12, August, 2012. Article (CrossRefLink).

[7] Basso, Simone, Michela Meo, and Juan Carlos De Martin, "Strengthening measurements from the edges: application-level packet loss rate estimation," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 45-51, July, 2013. Article (CrossRefLink).

[8] Silveira, Fernando, and Edmundo de Souza e Silva, "Predicting packet loss statistics with hidden Markov models for FEC control," *Computer Networks*, vol. 56, no. 2, pp. 628-641, February, 2012. Article (CrossRefLink).

[9] Hu, Zhiguo and Qiqiang Zhang, "A new approach for packet loss measurement of video streaming and its application," *Multimedia Tools and Applications*, pp. 1-20, May, 2016. Article (CrossRefLink).

[10] Nguyen, Hung X., and Matthew Roughan, "Rigorous statistical analysis of internet loss measurements," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 3, pp. 734-745, June, 2013. Article (CrossRefLink).

[11] NS-2 – The Network Simulator version 2.34, 2012, Article (CrossRefLink).

[12] Paxson, Vern, "End-to-End Internet Packet Dynamics," *ACM SIGCOMM Computer Communication Review*, vol.27, no.4, pp.139-152, September, 1997. Article (CrossRefLink).

[13] Ludwig R, Katz R H, "The Eifel algorithm: making TCP robust against spurious retransmissions," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 1, pp. 30-36, January, 2000. Article (CrossRefLink).

[14] Sarolahti, P., Kojo, M., Yamamoto, K., Hata, M, "An algorithm for detecting spurious retransmission timeouts with TCP," RFC 5682, IETF, September, 2009. Article (CrossRefLink).

[15] Rani, SV Jansi, and P. Narayanasamy, "Enhancing TCP Performance by detecting spurious RTO in Wireless Network," *International Journal of Applied Engineering Research*, vol.11, no.4, pp. 2651-2657, March, 2016. Article (CrossRefLink).

[16] Priya, S. Sathya, and K. Murugan, "Improving TCP Performance in Wireless Networks by Detection and Avoidance of Spurious Retransmission Timeouts," *Journal of Information Science and Engineering*, vol. 31, no.2, pp. 711-726, March, 2015. Article (CrossRefLink).

[17] Lan H, Ding W, Xia Z, "Asymmetric routing detection based on flow records," *Journal on Communications*, vol. 35, no. Z1, PP. 98-102, November, 2014. Article (CrossRefLink).

[18] Dainotti A, Benson K, King A, et al, "Estimating internet address space usage through passive measurements," *ACM SIGCOMM Computer Communication Review*, vol. 44, no.1, pp. 42-49, January, 2014. Article (CrossRefLink).

[19] Glatz E, Dimitropoulos X, "Classifying internet one-way traffic," in *Proc. of the 2012 ACM conference on Internet measurement conference*, pp. 37-50, November, 2012. Article (CrossRefLink).

[20] Miao L, Ding W, Gong J, "A real-time method for detecting internet-wide SYN flooding attacks,", in P*roc. of Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on. IEEE*, pp. 1-6, April, 2015. Article (CrossRefLink).

Haoliang Lan is a Ph.D candidate in school of computer science and engineering of Southeast University. His major research interests include network measurement, network management, and network security.

Wei Ding received B.S degree in the computer soft from Nanjing University in 1982. She received Ph.D degree from Southeast University in 1995. Nowadays she is a professor of Southeast University. Her major research interests include high speed communications, network management, and network security.

YuMei Zhang is a M.S candidate in school of big data and information engineering of Guizhou University. Her major research interests include network measurement and network management.