Every Packet Counts: Loss and Reordering Identification and Its Application in Delay Measurement

Jiliang Wang, *Member, IEEE, ACM*, Shuo Lian, *Member, IEEE*, Wei Dong, *Member, IEEE, ACM*, Xiang-Yang Li, *Fellow, IEEE*, and Yunhao Liu, *Fellow, IEEE, ACM*

***Abstract*—Delay is an important metric to understand and improve system performance. While existing approaches focus on aggregated delay statistics in pre-programmed granularity and provide results such as average and deviation, those approaches may not provide fine-grained delay measurement and thus may miss important delay characteristics. For example, delay anomaly, which is a critical system performance indicator, may not be captured by coarse-grained approaches. We propose a new measurement structure design called order preserving aggregator (OPA). Based on OPA, we can efficiently encode and recover the ordering and loss information by exploiting inherent data characteristics. We then propose a two-layer design to convey both ordering and time stamp, and efficiently derive per-packet delay/loss measurement. We evaluate our approach both analytically and experimentally. The results show that our approach can achieve per-packet delay measurement with an average of per-packet relative error at 2%, and an average of aggregated relative error at 10^{-5}, while introducing additional communication overhead in the order of 10^{-4} in terms of number of packets. While at a low data rate, the computation overhead of OPA is acceptable. Reducing the computation and communication overhead under high data rate, to make OPA more practical in real applications, will be our future direction.**

***Index Terms*—Delay measurement, fine-grained, loss and reordering.**

I. Introduction

A. Background

DELAY performance is of great importance for various network applications, ranging from daily used applications such as voice-over-IP, multimedia streaming, video on demand to applications in different areas such as data center and automatic trading system [1], [2] . For example, delay performance, which is closely related to the quality of service for applications such as voice-over-IP, significantly impacts user experience. Moreover, for delay sensitive system such as critical automatic trading, millions of trading may be conducted during a very short time period. Therefore, a small delay for operational packets may result in a significant impact on the trading amount and hence the profit [3]. Therefore, delay is a critical metric that system designers care about. It is of great importance to understand and improve system performance, and also has attracted a lot of research efforts [2], [4]–[9].

Intuitively, delay can be measured by embedding a sending time stamp in each packet. When a packet is received, the receiving time is recorded. The packet delay can thus be calculated by subtracting the sending time from the receiving time as long as the sender and receiver are synchronized. However, in most routers, it is difficult to modify the IP packets. Even if a time stamp can be inserted, this may require to add fields or use preserved fields in a packet [5]. Such modifications may not be aware by other protocols. Therefore, the behavior of other protocols may be affected.

To satisfy practical system requirements, a common applicable approach is to measure packet delay non-intrusively without modifying data packets. Following such a design principle, different methods are proposed [5]–[7] for providing aggregated delay statistics. For example, a representative method based on Lossy Difference Aggregator (LDA) is proposed for delay measurement [5]. With LDA, the aggregated delay for a group of packets can be calculated without modifications to packets.

B. Motivation

Existing approaches focus on providing aggregated delay statistics at a pre-programmed granularity. However, as revealed in existing approaches [10], fine-grained delay measurement is of great importance in performance monitoring, system diagnosis, traffic engineering, etc.

First, fine-grained delay measurement is important for revealing detailed system performance. Considering a simple example in which 999 packets have a delay of 1 ms and 1 packet has a delay of 1001 ms, this is different from the case in which all 1000 packets have a delay of 2 ms, though both cases exhibit the same aggregated average delay. Moreover, even for

Manuscript received August 18, 2014; revised January 29, 2015; July 09, 2015; and November 16, 2015; accepted January 14, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Fahmy. Date of publication February 08, 2016; date of current version December 15, 2016. This work was supported in part by the NSFC under Grants 61572277, 61529202, 61373166, and 61472360, the NSFC Major program under Grant 61190110, and the Fundamental Research Funds for the Central Universities.

J. Wang and Y. Liu are with the School of Software and TNLIST, Tsinghua University, Beijing 100084, China (e-mail: jiliangwang@tsinghua.edu.cn; yunhao@greenorbs.com).

S. Lian is with the School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710017, China (e-mail: lianshuo@mail.xjtu.edu.cn).

W. Dong is with the College of Computer Science, Zhejiang University, Hangzhou 310027, China (e-mail: dongw@zju.edu.cn).

X.-Y. Li is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: xli@cs.iit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2016.2523127

the same average and deviation, different types of packets may have different delay performance, e.g., large delays of interest may appear on a special set of packets that cannot be revealed from aggregated results on the entire set of packets.

Second, fine-grained delay performance is important to network diagnosis [10]. In diagnosis, it may need to investigate a special type of packets, e.g., DNS packet, ACK packet. For example, in a time-critical bidding system, the bidding request packet usually has a tight deadline [10]. Failure to meet the deadline may miss some bidding opportunity or even result in significant profit loss. Thus measuring per-packet delay of request packet is important in real time bidding system.

Third, fine-grained delay measurement is required or presumed in various protocols. It can also be used to improve protocol performance. For example, many protocols in Internet and data center [11]–[14] show that incorporating fine-grained delay measurement would improve the system performance.

Last but not least, inherently aggregated delay performance cannot be accurately calculated in the presence of packet losses or reorderings [5]–[7]. Even with a single lost packet in a group, the entire group has to be discarded and the average delay for such a group cannot be calculated. From system management perspective, packets with losses or reorderings should be particularly important to understand system behavior and reliability.

C. Our Approach

We propose a fine-grained delay measurement approach based on loss and reordering identification. We present a new data structure named Order Preserving Aggregator (OPA). In OPA, packet loss as well as ordering information can be efficiently represented and identified. The OPA design leverages the intrinsic data property that lost and reordering packets are usually much less than legitimate packets, which facilitates efficient ordering and loss information representation and recovery.

Based on OPA, we present a two-layer delay measurement design in which ordering and loss information, and time stamp information are separately transmitted and recovered according to their inherent properties. Then two layers of information are combined at the receiver to achieve fine-grained delay measurement.

Compared with existing approaches, OPA has several merits. First, OPA exploits the inherent data properties, and incurs a low computation and communication overhead for loss and reordering identification. Second, OPA can provide per-packet delay and loss measurement instead of aggregated statistics. Third, packet delays in groups with losses or reorderings, which is important but cannot be measured in existing approaches, can be derived.

The contributions are summarized as follows.

- *Architecture for loss and reordering identification*. We propose the OPA design, a new measurement structure to efficiently represent and identify the ordering and loss by exploiting intrinsic data properties. By applying the OPA design, we design a two-layered information representation system for fine-grained delay measurement.

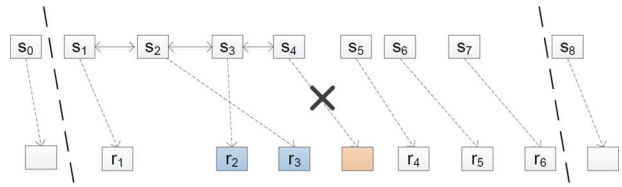


Fig. 1. Fine-grained delay measurement with packet loss and reordering.

- *Performance analysis*. We analyze the computation overhead at the sender and receiver, and the communication overhead for the proposed approach.
- *Performance evaluation*. The evaluation results demonstrate the effectiveness of the OPA approach. With an overhead in the order of 10^{-4} with respect to the total data packets, per-packet delay can be measured with an average relative error at 2%, and the aggregated delay can be measured with a relative error at 10^{-5}.

The remainder of this paper is organized as follows. Section II describes the assumptions and the network model. Section III introduces existing approaches. Section IV introduces the design of OPA. Section V shows how to leverage OPA for per-packet delay measurement. Section VI shows the analysis result and Section VII shows the evaluation results. Section VIII emphasizes the limitations on computation and possible future work. Section IX concludes this work.

II. Assumptions and Network Model

We consider measuring packet delay from a $sender$ to a $receiver$. For example, a sender and receiver can be two routers in a network. We divide packets into segments. We use two packets as *delimiter packets* for the sender and receiver to agree on the start and end of each segment. For example, as shown in Fig. 1, packets from the sender are s_1, s_2, …, s_n. The delimiter packets are s_1 and s_n, which can be used by the receiver to locate the corresponding segment, i.e., r_1, r_2, …, r_m where $r_1 = s_1$ and $r_m = s_n$. Without packet loss and reordering, we should have $m = n$ and $s_i = r_i$ for $1 \leq i \leq n$.

In practice, there may exist packet loss or packet reordering. As a result, the sending ordering of packets is not necessarily the same with the receiving ordering of packets. For example, as shown in Fig. 1, packet s_2 is a reordering packet and packet s_4 is a lost packet. Delimiter packets may also get lost. If a delimiter packet is lost, we discard the corresponding segment and move to the next segment. Hereafter we assume delimiter packets are successfully received.

As in previous approaches [5]–[7], we assume that the sender and receiver are synchronized. This can be achieved by existing time synchronization protocols [15], [16] . We assume there is no common sequence number since packets may come from different sources with different protocols [5], [7] . Meanwhile, packets are not to be modified. This is common for the Internet routing infrastructure, in which the intermediate routers do not modify the packet.

For each packet, the sender can measure the *sending time* t_i^s. When packet r_i is received, the receiver can measure the *receiving time* t_i^r. Our goal is to calculate the delay for each

received packet. The delay is defined as the receiving time subtracted by the sending time. For example, if packet r_i corresponds to s_j at the sender, the delay can be calculated as $t_d(i) = t_i^r - t_j^s$, where t_i^r and t_j^s are the receiving time and sending time respectively. Hereafter, we focus on per-packet delay measurement for a segment of n packets.

III. Existing Approaches

A. Timestamping Based Approaches

A straightforward approach for delay measurement is to insert a time stamp in each packet. We call such kind of method timestamping (ts) based method. However, ts based method requires modifications to packets, which is not applicable on commonly used routers. Even packets can be modified, this incurs additional transmission cost. Thus inserting time stamps is not preferable for practical applications.

B. Probing Based Approaches

Probing is a commonly used technique to estimate packet loss and delay. In probing based methods, probing packets are sent from the sender to the receiver. Those probing packets are assumed to have similar behavior with other packets. Based on probing packets, the statistics for other packets can be estimated. Probing based methods, which significantly reduce the measurement overhead, fail to achieve fine-grained delay measurement [5]–[7].

C. LDA

Loss Difference Aggregator (LDA) is proposed to estimate delay average and deviation. In LDA, packets are divided into groups by a certain hash function. For each group, the sum of the time stamps and the total counter of packets are sent to the receiver. Upon receiving such information, the receiver first divides packets by applying the same hash function. If the total packet counter for the group at the receiver matches that at the sender, the receiver can calculate the average delay for such a group. Otherwise the receiver will discard such a group.

The average delay is calculated as follows. The sum of delays can be calculated by subtracting the sum of sending time stamps from the sum of receiving time stamps. Then the average delay can be calculated by dividing the sum of delay by the packet counter. Therefore, LDA significantly reduces the measurement overhead by only transmitting the sum of time stamps and packet counter. However, when there exist packet loss and reordering in a group, the entire group has to be discarded and the delay for such a group cannot be estimated. With even a single loss, the entire group becomes useless.

D. FineComb

In LDA, packets that belong to one segment may be misidentified into other segments due to packet reordering. After dividing a segment of packets into groups, some groups may have packets from other segments. In such a case, those groups of packets cannot be used for delay measurement. To address such a problem, FineComb [7] proposes a special data structure called stash, which maintains the information for packets near the boundary of segments at the receiver. However, for groups with lost packets, FineComb still cannot calculate their per-packet delay.

TABLE I
Comparison of Existing Approaches

methods	time-stamping	probe	per-packet delay	deal with loss and reordering	abnormal delay detection
ts based	Yes	No	Yes	Yes	Yes
probe based	No	Yes	No	Yes	Yes
LDA	No	No	No	No	No
FineComb	No	No	No	Partially	No
RLI	No	Yes	No	Yes	No
MAPLE	Yes	No	Yes	Yes	Yes
OPA	No	No	Yes	Yes	Yes

E. RLI

A per-flow delay measurement approach called Reference Latency Interpolation (RLI) is proposed in [6]. In RLI, the sender generates reference packets (which are similar to probe packets) to the receiver. Then based on reference packets, RLI uses interpolation to estimate per-packet delay between reference packets. The interpolation based method inherently assumes a specific delay distribution (e.g., linear delay distribution). Thus it may not be able to accurately measure the per-packet delay in the presence of frequent or significant delay variations. For example, a sudden increase of packet delay between two reference packets cannot be captured, resulting in missing of important information to investigate delay variations or anomalies.

F. MAPLE

Recently, MAPLE [10] is proposed to store and query per-packet delay. MAPLE consists of two main components: a scalable packet latency store (PLS) and a query engine. PLS can be used to efficiently store the calculated delay and the query engine can be used to query packet delay. MAPLE focuses on delay storage and query rather than per-packet delay measurement. We focus on packet delay measurement. MAPLE can be further leveraged as the storage and query system for fine-grained delay measured in our method.

Summary: We summarize existing approaches for delay measurement in Table I. We can see that none of those approaches can achieve per-packet delay measurement in the presence of loss and reordering. Therefore, we propose OPA to achieve efficient per-packet delay measurement in the presence of loss and reordering without timestamping and probing.

IV. OPA Design

In this section, we present the OPA design. The design goals of loss and reordering measurement and delay measurement are as follows.

- The design should be non-intrusive and should not require any modification to data packets. Thus the design does not introduce any change to existing routing protocols.
- The design should be light-weight and efficient, not incurring much additional computation and communication overhead.
- The abnormal delay, e.g., large delays, should be captured and preserved in order to investigate important system metrics.

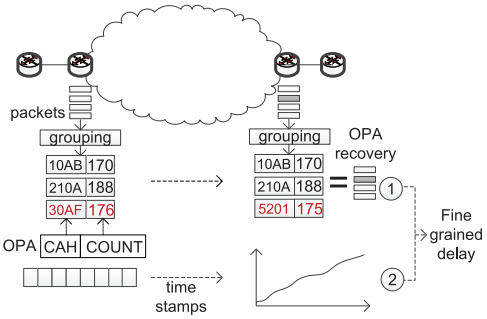


Fig. 2. The design overview of OPAs for fine-grained delay and loss measurement.

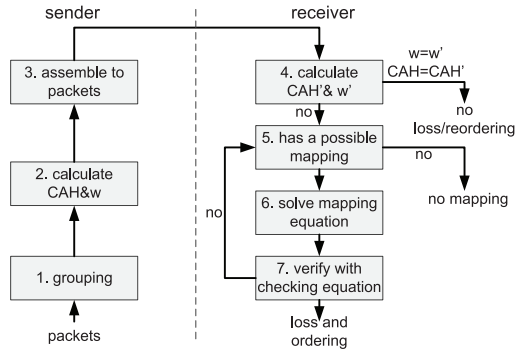


Fig. 3. The working flow for loss and ordering information recovery.

A. Design Overview

A straightforward approach is to send time stamp information from the sender to the receiver. More specifically, we attach a packet ID to each time stamp in order to calculate per-packet delay at the receiver side. Moreover, an immediate improvement is to send the compressed time stamps to the receiver. At the receiver side, the time stamps can be recovered and the delay can be calculated given the receiving time stamps. However, this does not work in the presence of packet loss and reordering. For a single packet loss or reordering, the compressed information cannot be used since the receiver does not know which one is lost. As shown in Fig. 1, assume all packets are correctly received except packet s_4. Since the receiver does not know which packet is lost, it incorrectly calculates the sending time of packet r_5 as t_5^s (packet r_5 should correspond to s_6). What is even worse is that a single packet loss would pollute an entire group of packets. Thus the aggregated delay performance (e.g., average delay) cannot be calculated, not even say per-packet delay. This problem is exacerbated when there exists packet reordering.

Our basic idea is to efficiently recover the ordering information at the receiver. The ordering information corresponds to the position for each packet at the sender. It is challenging to represent the ordering information since there are a large number of possible orderings for the received packets. However, we find that the receiving ordering should be very similar to the sending ordering. In production networks, the number of reorderings and lost packets is usually small. Intuitively, the ordering "difference" between the sender and receiver should be small. Thus instead of transmitting information to represent all possible orderings, we consider transmit the small ordering difference to reduce the overhead.

As shown in Fig. 2, we propose a two-layer representation design. Packets are first divided into groups at the sender side. For each group, the sender calculates the corresponding OPA (the first layer) and the compressed time stamps (the second layer), and then transmit the two-layer information to the receiver. The receiver can efficiently recover packet ordering with OPAs. Then those two layers are combined for per-packet delay measurement. We introduce the first layer in Section IV and the second layer in Section V.

B. Building Order Preserving Aggregator

The main steps to recover ordering information are shown in Fig. 3. At the sender, the first step is to divide packets into groups based on a deterministic hash function $Hash(\cdot)$. The hash function maps any string to an integer in the range $[1, g]$. Therefore, packets in a segment are divided into g groups, i.e., $G_1, G_2, \ldots, G_g$. For brevity, we assume each group has w packets, i.e., $n = g \times w$. By using a hash function, bursty packet losses (e.g., due to congestion or buffer overflow) can be divided into different groups. To simplify the presentation, we denote packets in group G_i as $G_i = (s_{i1}, s_{i2}, \ldots, s_{iw})$, where s_{ij} is the jth packet in group G_i.

The second step is to represent packet ordering. We design a method named augmented hash to encode the ordering. For group G_i, the sender calculates the augmented hash value of the jth packet s_{ij} as $h_{ij} = Hash(s_{ij} + +j)$, where $++$ can be a string concatenation operation. We call h_{ij} as the augmented hash of the jth packet in group G_i. We denote H_i as the augmented hash vector as $H_i = (h_{i1}, h_{i2}, \ldots, h_{iw})^T$, where $(H_i)^T$ denotes the matrix transpose of H_i. For group G_i, the sender calculates the augmented hash values for all packets. If the receiving ordering is exactly the same with the sending ordering, the augmented hash values should be exactly the same.

To further reduce the overhead, we calculate the linear combinations of the augmented hash values for each group. For group G_i with w packets, the sender calculates k linear combinations $B_i = (b_{i1}, b_{i2}, \ldots, b_{ik})^T$ as follows

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1w} \\ a_{21} & a_{22} & \ldots & a_{2w} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \ldots & a_{kw} \end{pmatrix} \begin{pmatrix} h_{i1} \\ h_{i2} \\ \vdots \\ h_{iw} \end{pmatrix} = \begin{pmatrix} b_{i1} \\ b_{i2} \\ \vdots \\ b_{ik} \end{pmatrix}. \quad (1)$$

Here $A = (a_{xy})_{1 \le x \le k, 1 \le y \le w}$ is a coefficient matrix. We explain how to construct the coefficient matrix in the Appendix. We call the vector $B_i = (b_{i1}, b_{i2}, \ldots, b_{ik})^T$ for group G_i the *Combined Augmented Hash* (CAH). The CAH can be calculated by the sender. Equation (1) can also be written as

$$A \times H_i = B_i. \quad (2)$$

It should be noted that B_i has k values and H_i has w values. For group G_i, the sender calculates the combined augmented hash B_i and then sends a tuple (B_i, w) to the receiver, where B_i is the CAH and w is the number of packets in group G_i. We call such a tuple the *Order Preserving Aggregator* (OPA).

In the third step, the sender calculates the OPAs for all groups. Then the sender assembles tuples for multiple groups into packets and then send the packets to the receiver.

C. Identify Correct Groups

To recover the ordering information, the receiver performs the following steps. The receiver divides packets into groups using the same hash function (i.e., $Hash(\cdot)$) with that at the sender. Assume the received groups are $G'_1, G'_2, \ldots, G'_g$. We denote packets in G'_i as $G'_i = (r_{i1}, r_{i2}, \ldots, r_{iw'})$. As in the 4th step in Fig. 3, we also calculate the augmented hash for each received packet r_{ij} as $h'_{ij} = Hash(r_{ij}++j)$. The corresponding result is $h'_i = (h'_{i1}, h'_{i2}, \ldots, h'_{iw'})^T$. For group G_i, the received $OPA_i = (B'_i, w')$. Based on the received and calculated information, there are two cases for a group G'_i.

- $w' = w$ and $B_i = B'_i$: the group of packets in G'_i should be the same with the packets in G_i at the sender. Based on $A \times H_i = B_i$, the value of H_i that satisfies the equation should be on the intersection of hyperplanes. As packets are randomly lost, the probability that the calculated H'_i is on the intersection of k hyperplanes is negligible. Meanwhile, since we use augmented hash for calculating CAH, the receiving ordering should also be same with sending ordering. Based on the OPAs, we can check if the packet ordering is preserved at the receiver, and we can also check if the received group is exactly the same with the sending group.
- Otherwise, those two groups of packets are not the same, which indicates the existence of packet losses or packet reorderings from group G_i to G'_i.

D. OPA Recovery

Then we deal with the case with packet loss or packet reordering. First, we consider the case with packet loss. For presentation simplicity, assume there are $\alpha = w - w'$ packet losses. Packet reordering can be processed similarly. Later, we explain how to deal with the case with both packet losses and reorderings. When α is less than k, we show that OPA can be used to derive the ordering information and lost positions.

It is difficult to derive the position for those lost packets, since we do not know the correct position for those w' received packets in a group and we do not know which α packets are lost. Due to packet losses, the receiver may not have enough augmented hash values to calculate $\mathrm{CAH}'(B'_i)$. Our goal is to determine the position of lost packets and the correct ordering of the received w' packets.

More specifically, we need to determine the ordering and position for the received w' packets among the w packets from the sender, i.e., mapping the receiving w' packets to the correct positions. Before diving into details, we first show the intuition of determining the position and ordering for the received w' packets. Intuitively, as there are α lost packets, we assume there are α unknown variables and enumerate the position for those α variables. This corresponds to the 5th step in Fig. 3. For each enumeration, we calculate the augmented hash values for those received packets. Remember that we have $A \times H_i = B_i$, using the first α equations, we can solve the equation set and obtain the result for α variables, i.e., the augmented hash values for lost packets. This corresponds to the 6th step in Fig. 3. Then the solved augmented hash values for lost packets and received packets are checked with the remaining equations. If the positions for lost packets are correct, the solved augmented hash values should satisfy the remaining equations. This is the 7th step in Fig. 3.

We first show an example of the proposed method. Then we explain the details behind the intuition.

Example: We use a simplified example to explain the basic procedure of our approach. Assume a group G_1 has three packets (s_{11}, s_{12}, s_{13}), say (2, 5, 7). We use a very simple augmented hash function as $h_{ij} = Hash(s_{i,j++j}) = s_{i,j++j}$.

The sender performs the following steps:

- The sender calculates the augmented hash values for those three packets: $H_1 = (h_{11}, h_{12}, h_{13})^T = (21, 52, 73)^T$;
- Assume the coefficient matrix $A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$, the sender calculates the CAH as

$$B_1 = AH_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 21 \\ 52 \\ 73 \end{pmatrix}.$$

  Therefore, $B_1 = (1{\cdot}21{+}1{\cdot}52{+}1{\cdot}73, 1{\cdot}21{+}2{\cdot}52{+}3{\cdot}73) = (146, 344)$;
- The sender calculates the OPA as ((146, 344), 3), where $w = 3$.

The receiver performs the following steps:

- The receiver receives OPA $= ((146, 344), 3)$.
- If the received packets are $G'_1 = (2, 5, 7)$, the receiver can calculate $H'_1 = (21, 52, 73)$.
- The receiver can calculate $B'_1 = AH'_1 = (146, 344)$ and $w' = 3$.
- The receiver have $\mathrm{OPA}' = ((146, 344), 3) = \mathrm{OPA}$.

Thus from $OPA = OPA'$, we know there is no loss and reordering. If the received two packets are $r_{11} = 2$, and $r_{12} = 7$, the second packet 5 is lost and the other two packets are received. In such a case, The receiver performs the following steps:

- The receiver finds $\mathrm{OPA}' \neq \mathrm{OPA}$.
- The receiver calculates the number of lost packets as $\alpha = w' - w = 1$.
- The receiver enumerates the lost positions. Assume the first packet is lost, i.e., we have $(x, 2, 7)$. According to the augmented hash function, we have $h'_{12} = 22$ and $h'_{13} = 73$. Thus we have

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} h'_{11} \\ h'_{12} \\ h'_{13} \end{pmatrix} = \begin{pmatrix} 146 \\ 344 \end{pmatrix}.$$

  By $1{\cdot}x{+}1{\cdot}h'_{12}{+}1{\cdot}h'_{13} = 146$, we have $x = 51$. Using this for the checking equation, we have $1{\cdot}x{+}2{\cdot}h'_{12}{+}3{\cdot}h'_{13} = 314 \neq 344$. Thus the lost packet is not the first packet. Similarly, we can check whether the second packet is lost. Finally, we identify the second packet as the lost packet.

E. Mapping and Checking Equation

In this section, we explain the details for recovering the ordering information. Table II summarizes important parameters used in this paper. We denote $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\beta)$ as a mapping vector. The jth element σ_j denotes that the jth packet in

TABLE II
NOTATIONS

parameter	description
A	The coefficient matrix.
B_i	Combined augmented hash value.
w	# of packets in group G_i.
OPA_i	Order preserving aggregator, i.e, (B_i, w).
σ	A mapping from received packets to sent packets
A_σ	The matrix after applying σ-mapping to A.
H_σ	The combined augmented hash values calculated from σ-mapping for received packets.
$A_{\bar{\sigma}}$	The resulted matrix by remove A_σ from A.
$H_{\bar{\sigma}}$	An unknown vector that needs to be solved.
α	# of packet losses
g	# of groups
d	reordering shift
δ	# of total errors
ξ	# of errors in a group
ρ	packet loss rate
$t_d(i)$	delay for packet i
$t_d^F(i)$	estimated delay for packet i
η	average delay error
ϵ_i^F	delay error of packet i for method F

G'_i is the σ_j packet in G_i. $\sigma_j = 0$ means that the jth packet in G'_i has no corresponding packet in G_i. We have the following definition.

Definition 1: For $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{w'})$, we define a σ-mapping from a vector $v = (v_1, v_2, \ldots, v_{w'})$ to a vector of length w as $v_\sigma = (v_{j_1}, v_{j_2}, \ldots, v_{j_w})$ where $v_\sigma(i) = v_{j_i}$ if there is an index j_i such that $\sigma_{j_i} = i$, otherwise, $v_\sigma(i) = \emptyset$.

By performing σ-mapping for w' packets in G'_i, we obtain a new ordering for packets. For example, if the received packets $v = (v_1, v_2, v_3)$ and $\sigma = (2, 3, 1)$, we have $v_\sigma = (v_3, v_1, v_2)$. If $v = (v_1, v_2, v_3)$ and $\sigma = (2, 3, 0)$, we have $v_\sigma = (\emptyset, v_1, v_2)$.

With the σ-mapping from the received w' packets to w packets in G_i, according to (2), we have

$$\begin{aligned}
&\sum_{\sigma_j \neq 0} (a_{1,\sigma_j} \cdot Hash(r_{ij} + +\sigma_j)) + \sum_{\nexists_j \sigma_j = l} a_{k,l} \cdot h_l = b_{i1} \\
&\sum_{\sigma_j \neq 0} (a_{2,\sigma_j} \cdot Hash(r_{ij} + +\sigma_j)) + \sum_{\nexists_j \sigma_j = l} a_{k,l} \cdot h_l = b_{i2} \\
&\qquad \ldots \\
&\sum_{\sigma_j \neq 0} (a_{k,\sigma_j} \cdot Hash(r_{ij} + +\sigma_j)) + \sum_{\nexists_j \sigma_j = l} a_{k,l} \cdot h_l = b_{ik}. \quad (3)
\end{aligned}$$

The first part $\sum_{\sigma_j \neq 0}(\cdot)$ in the equation is to calculate the combined augmented hash calculated by mapping the received packets to the correct position. The second part $\sum_{\nexists_j \sigma_j = l}(\cdot)$ in the equation is to calculate the combined augmented hash for lost packets.

Denote the matrix $A_\sigma = (a_{i,\sigma_j})$ for $1 \leq i \leq k$, $1 \leq j \leq w'$ and $\sigma_j \neq 0$. Denote the column vector $H_\sigma = (Hash(r_{ij} + +\sigma_j))^T$ for $1 \leq j \leq w'$ and $\sigma_j \neq 0$. Denote the matrix $A_{\bar{\sigma}} = (a_{i,l})$ where $\nexists_j \sigma_j = l$ and $1 \leq i \leq k$. Denote $H_{\bar{\sigma}}$ as a unknown column vector of length $w - w'$. Equation (3) can be written as

$$A_\sigma \times H_\sigma + A_{\bar{\sigma}} \times H_{\bar{\sigma}} = B_i. \quad (4)$$

We have no information for H_σ since we do not know the ordering of the received packets. Therefore, both H_σ and $H_{\bar{\sigma}}$ are unknown in (4). Meanwhile, we do no know the mapping vector σ of the received packets.

Algorithm 1 OPARecovery

1: $\sigma \leftarrow 0$;
2: Calculate B'_i from the received OPA for group G'_i;
3: **for all** possible σ **do**
4: Calculate H_σ, A_σ;
5: Calculate $H_{\bar{\sigma}}$ according to mapping (5);
6: **if** The checking equation (6) is satisfied **then**
7: **return** (σ, $\bar{\sigma}$);
8: **return** (NULL, NULL);

To obtain the value of H_σ and $H_{\bar{\sigma}}$, we leverage the properties of packets in G_i and G'_i, to first find the value of σ. To find the value of σ, we consider different cases for (4). First, if there are no losses and reorderings, we have $w' = w$ and $\sigma = (1, 2, \ldots, w)$. In this case we can easily obtain H_σ and $H_{\bar{\sigma}}$. Second, if there are some losses and reorderings, we enumerate all possible packet losses and reorderings. For each enumeration, we can obtain a mapping σ to solve (4). Usually, the number of lost packets should be much less than that of received packets.

When there are α packet losses, we have $|\bar{\sigma}| = \alpha$. Meanwhile, A_σ has $w - \alpha$ columns. For any mapping vector σ, we can calculate H_σ based on the received packets. Thus we have the following *mapping equation*:

$$A_{\bar{\sigma}}[1, \alpha] \times H_{\bar{\sigma}}[1, \alpha] = B_i[1, \alpha] - A_\sigma[1, \alpha] \times H_\sigma[1, \alpha] \quad (5)$$

where $A_{\bar{\sigma}}[1, \alpha]$ is the matrix consisting of row 1 to row α of $A_{\bar{\sigma}}$. If any α columns from $A_{\bar{\sigma}}[1, \alpha]$ are independent, we can solve the mapping equation and obtain the value for $H_{\bar{\sigma}}$. After solving the mapping equation, we still do not know whether the solution is a feasible solution since we do not have the ground truth for $H_{\bar{\sigma}}$. Therefore, we leverage the remaining rows in matrix A to verify the feasibility of the solution of $H_{\bar{\sigma}}$.

To verify the solution, we require $\alpha < k$. We use the remaining $k - \alpha$ rows as checking rows. Accordingly, we have the *checking equation*:

$$\begin{aligned}
&A_{\bar{\sigma}}[\alpha + 1, k] \times H_{\bar{\sigma}}[\alpha + 1, k] \\
&\qquad = B_i[\alpha + 1, k] - A_\sigma[\alpha + 1, k] \times H_\sigma[\alpha + 1, k]. \quad (6)
\end{aligned}$$

The solution is feasible only when it satisfies the checking equation. In summary, the mapping equation is used to find possible mappings. The checking equation is used to verify the feasibility of each mapping. When all results for the mapping equation cannot be satisfied with the checking equation, the corresponding group cannot be used.

Algorithm 1 illustrates the main steps to recover the ordering information. We enumerate possible σ to calculate H_σ. Based on H_σ and the mapping equation (5), we can obtain $H_{\bar{\sigma}}$. Then we can check $H_{\bar{\sigma}}$ whether it is feasible with the checking equation (6). Line 2 is to calculate CAH from the received OPA for group G_i. Line 4 to line 5 show the calculation of $H_{\bar{\sigma}}$. Line 6 shows using the checking equation to verify $H_{\bar{\sigma}}$.

Algorithm 2 CalculateDelay

1: $(\sigma, \bar{\sigma}) \leftarrow OPARecovery()$;
2: Record the received time stamps $t_1^r, t_2^r, \ldots, t_{w'}^r$;
3: Recover the sending time stamps $\tilde{t}_1^s, \tilde{t}_2^s, \ldots, \tilde{t}_w^s$;
4: $t_d(i) \leftarrow -1$ for $1 \leq i \leq w'$; //initialization
5: **for** $i = 1$ to $|\sigma|$ **do**
6: **if** $\sigma(i) \neq 0$ **then**
7: $t_d(i) \leftarrow t_i^r - \tilde{t}_{\sigma(i)}^s$;

V. Delay Calculation

After recovering the positions for all received packets with OPA, we then calculate the per-packet delay.

A. Time Stamp Recovery

To calculate fine-grained delay, we send the compressed time stamps to the receiver in the second layer. The variation of each time stamp will not be large since packets are sent on a high speed data link. As we have recovered the ordering/loss information in the first layer, here we can use lossy compression method to achieve a high compress ratio. For example, we can use wavelet for time stamp compression.

In wavelet compression, the error for all recovered time stamps is nearly zero-sum. Thus the average delay calculated is very closed to the true average. The error is evenly distributed and very small. It is worth noting that time stamp compression is not the focus of our method and other compression methods can also be used.

Denote the recovered time stamps as $\tilde{t}_1^s, \tilde{t}_2^s, \ldots, \tilde{t}_w^s$. For each packet, the receiver can also record the receiving time stamps $t_{i_1}^r, t_{i_2}^r, \ldots, t_{i_{w'}}^r$. According to the result of σ, we can calculate the corresponding position for the received packets. Thus we can calculate the delay for those w' packets as $t_d(i) = t_i^r - \tilde{t}_{\sigma(i)}^s$, where $\sigma(i)$ is the ith element in σ and $\sigma(i) \neq 0$. The main steps to calculate delay are shown in Algorithm 2. Line 2 and Line 3 show the steps to obtain the sending and receiving time stamps. Line 5–7 show how to use the mapping vector to calculate the delay for each packet.

B. Deal With Reordering

In the presence of reordering, we need to accordingly enumerate possible mapping vector σ. For example, for a group of four packets $(1, 2, 3, 4)$. If the received packets are $(1, 3, 2)$. We have $\sigma = (1, 3, 2)$.

Intuitively, we need to check all possible reorderings and generate corresponding mapping vector σ. Practically, for a received packet r_x, the shift between the original position and the reordered position is assumed to be bounded by d [10]. Thus in the mapping vector σ, the possible positions are $x-d, x-d+1, \ldots, x, x+1, x+d$. After grouping, the possible positions for a reordering packet are reduced. For example, with 10 groups, the number of possible positions for a packet is reduced from d to $d/10$ after grouping. For groups with reordering shift larger than d, the solved results for the mapping equation cannot be satisfied with the checking equation.

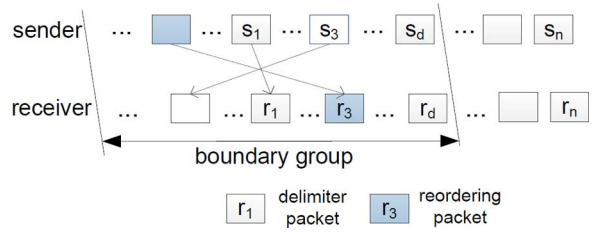


Fig. 4. Maintain information near the delimiter packets.

C. Deal With Boundary

Packet reordering inside a segment can be recovered by aforementioned method. Due to packet reordering near the boundary, packets from one segment (segment $k-1$) may move into neighboring segment (e.g., segment k) [7]. In such a case, packets from segment $k-1$ will be incorrectly considered as packets in segment k.

For example, as shown in Fig. 4, packet r_3 is not the same with the packet s_3 from the sender. Due to packet reordering, a packet from another segment moves into the middle segment. If such a packet is not identified, it will be counted for the middle segment and result in errors in delay measurement. We maintain a stash [7] for packets near the boundary. The size of the stash can be $2d$ in order to contain all possible reorderings. We can keep the packets of the stash at the receiver. Only the time stamps need to be sent from the sender to the receiver.

D. Discussion

It is possible that different groups in a segment have different sizes. In case that some group has more than w packets, the sender can extend the columns of the coefficient matrix A. As long as matrix A has more than w columns, the proposed method can still be used.

To achieve fine-grained delay measurement, it is required to maintain a buffer for all receiving time stamps. Assume there are n packets in a segment and we need $4n$ bytes to store the time stamps. For g groups with each group of s bytes, the total memory size is $4n + s \times g$. For example, for a segment of 10^6 packets, the required memory can be calculated as $4 \times 10^6 + 28 \times 2 \times 10^5$ when the number of group is 2×10^5 and s is 28. The total memory requirement is 9.6×10^6. This is acceptable when the size of each segment is small. However, when the size of segment increases, e.g., 10^8, the required memory will increase to 9.6×10^8, which is very large. Our approach can also be used in a on-demand manner. A user can specify particular constraints on the packets to be measured. For example, a user can measure packets in a particular flow or in a particular time period. In such cases, the information to be maintained is further reduced.

Considering the computation overhead and information representation of OPA, the received information can also be stored on the receiver. The stored information can then be used offline to calculate per-packet delay .

It is possible that packet loss may not be random, e.g., packet loss may be bursty due to congestion. The hash function will divide packets into groups. Thus bursty losses will be divided

into different groups. The distribution of packet losses will not affect the effectiveness of our approach.

Our approach leverages the property that the number of packet errors should be much less than the number of legitimate packets. When the packet error rate is very high (e.g, 1/10), OPA will increase the number of packet groups. The communication overhead accordingly increases due to increasing of groups. Consequently, the benefit of OPA decreases.

VI. Analysis

We seek to answer the following questions through analysis.

- What is the overhead and limitations for the proposed approach?
- What is the performance to identify the losses and reorderings?
- How to determine different parameters for the proposed approach?

A. Computation Overhead

The main computation overhead is to enumerate all positions of lost packets. As seen from Algorithm 1 and Algorithm 2, the overhead is determined by the number of errors and groups. With more groups, the expected number of errors in each group will become smaller and thus the computation overhead for each group will decrease. Meanwhile, the number of OPAs and the information to the receiver will increase. Therefore, we should carefully choose the group number to tradeoff the computation overhead and communication overhead.

For the computation overhead, we first calculate the probability for different number of errors in each group. Assume there are δ errors, and n packets are divided into g groups with each group having $w = \frac{n}{g}$ packets.

Denote $pr(err = \xi)$ as the probability that a particular group has ξ errors, we have

$$pr(err = \xi) = C(\delta, \xi) \cdot (\frac{1}{g})^{\xi} \cdot (1 - \frac{1}{g})^{\delta - \xi}. \quad (7)$$

Those ξ packet losses can be distributed to $C(w, \xi)$ positions in a group. Thus the number of possible permutations we need to check for each group is $C(w, \xi)$. Further, for ξ packet reordering, the number of possible permutations is $d \cdot C(w, \xi)$, where d is the maximum shift for a reordering packet. The shift can be reduced to d/g after dividing packets into g groups.We first consider packet loss. The expected overhead can be calculated as

$$E(overhead) = \sum_{\xi=0}^{\delta} C(w, \xi) \cdot pr(err = \xi) \cdot g. \quad (8)$$

In practice, we select the number of groups proportional to the number of errors, e.g., we can select the number of groups as $10\delta \sim 20\delta$. When the number of errors is small, the number of groups is relatively small. Since the number of errors is usually very small in each group, we ignore groups with errors more than a threshold in order to further reduce the overhead. Accordingly, we calculate the expected overhead $E(overhead')$ as

$$E(overhead') = \sum_{\xi=0}^{\alpha} C(w, \xi) \cdot pr(err = \xi) \cdot g \quad (9)$$

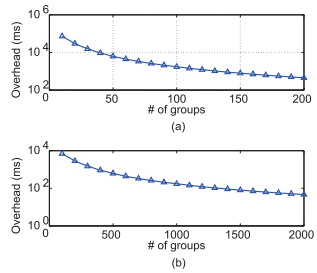


Fig. 5. Computation overhead with respect to the number of groups. (a) $\rho = 10^{-5}$ and (b) $\rho = 10^{-4}$.

where α is the maximum number of errors in a group. We can also see that for a larger α, there will be more recovered packets. Meanwhile, the computation overhead accordingly increases. Fig. 5 shows the computation overhead for different number of groups with different packet loss rates. The total number of packets is 10^6. The loss rate is 10^{-5} for Fig. 5(a) and 10^{-4} for Fig. 5(b). It can be seen that for both loss rates, the overhead decreases as the number of groups increases. This is because the group size and number of errors in each group decrease as the number of groups increases. With more groups, the computation overhead will further decrease as shown in Fig. 5.

The computation overhead at the sender is mainly due to OPA calculation. Each OPA requires multiplications $\frac{n}{g}$ times. For a group with k OPAs, the computation overhead is calculated as $\frac{n}{g} \times k$. The total overhead for n packets is $g \times \frac{n}{g} \times k = nk$, which is proportional to the number of packets.

In practice, a router should not solely work either as a sender or receiver, but to work as both of them. When the data rate is high, packets may not be able to be processed in real time due to the computation overhead. Considering the limitations, OPA can be more appropriate for a particular set or a particular flow of packets, or be used in an on-demand manner on required routers rather than on all routers all the time. In those scenarios, the overhead can be reduced.

B. Communication Overhead

We also consider the communication overhead from the sender to the receiver. The communication overhead consists of two parts according to the two-layer design in Fig. 2. The first part is the overhead for sending OPAs and the second part is for sending compressed time stamps. For the first part, the overhead for each group is calculated as the size of B_i plus the size of w. If CAH of B_i is of length 8 and w is of length 4, the total communication overhead for OPAs is $3 \times 8 + 4 = 28$ for a group with no more than 2 errors. For the second part, the overhead depends on the compression ratio. The overhead can be calculated as θL_2 where θ is the compression ratio and L_2 is the number of time stamps. Thus the total overhead is $28 + \theta L_2$ for each group.

Assume there are 100 groups for a segment of 10^6 packets, $\theta = 1/10$ and a time stamp has 4 bytes, the total overhead is 402800 bytes, which can be assembled in about 300 packets (with MTU $= 1500$ bytes). The amortized overhead is about $3/10^4$. This also means for 10^4 packets, no more than 3 packets are required to achieve fine-grained delay measurement.

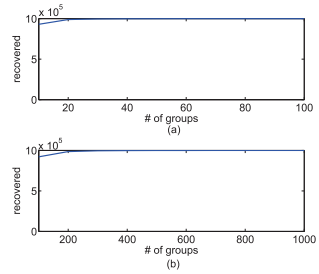


Fig. 6. Recovered delay with respect to the number of groups. (a) $\rho = 10^{-5}$ and (b) $\rho = 10^{-4}$.

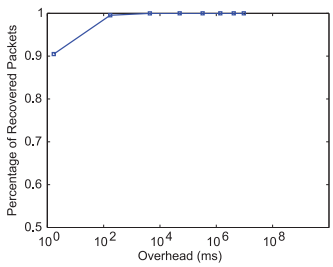


Fig. 7. Computation overhead with respect to recovered packets. The number of packets is 10^6.

C. Storage Overhead

RLI has the lowest storage size since it only needs to maintain reference packets. LDA has a small storage overhead for the time stamp sum and counter, which are proportional to the rows in all banks. The storage overhead of FineComb is that of LDA plus the storage for stash. Among those approaches, OPA has the highest overhead. It needs to maintain all OPAs which is proportional to the number of groups. It also needs a storage space for the compressed time stamps.

D. Recovered Delays

For a group with less than k errors, the probability it can be recovered is $pr(err < k)$. Therefore, the expected number of packets that can be recovered in a group is $pr(err < k) \times \frac{n}{g}$. Fig. 6 shows the number of recovered packet delays when we only consider groups with less than two errors ($k = 2$). We can see that with groups more than 10δ, e.g., 20 for a low error rate 10^{-5} and 200 for a high error rate, almost all packets can be recovered.

E. Impact of Parameters

The first parameter to consider is g. Intuitively, more groups indicate less errors in each group and thus less computation overhead. On the other hand, more groups result in more communication overhead. We require the expected overhead $E(overhead')$ for n packets is less than a threshold thr, i.e., $E(overhead') < thr$. We choose the smallest number of groups with overhead less than thr. Accordingly, we can calculate the number of recovered packets and the computation overhead for different k. Fig. 7 shows the percentage of recovered packets with respect to computation overhead. We can see that when $g = 10\delta$ and $k = 2$, most packets are recovered while the computation overhead is acceptable.

It should be noted that when the error rate is very high (e.g., 1/10), though increasing the number of groups can reduce the computation overhead, this inevitably increases the communication overhead and diminish the benefit of our proposed approach.

When the loss rate is unknown, we leverage a similar approach used in LDA and FineComb. We put packets to multiple banks for different loss rates. In each bank, we divide packets to groups according to its corresponding loss rate. Therefore, when OPA cannot solve the mapping equation for a bank, the packet loss ratio set for this bank is less than the real value and this bank cannot be used. When OPA can solve the equation for a bank, the packet loss ratio can be calculated and such a bank can be used for delay calculation. As in LDA, as long as there exists one bank that can be used, the delay can be calculated. In OPA, we do not maintain a disjoint set for different banks.

VII. Evaluation

A. Data

We use the Weibull delay distribution model to generate delay in our evaluation. The delay has a cumulative distribution function $pr(X \leq x) = 1 - e^{(-x/\lambda)^K}$ where λ and K denote the scale and shape of the distribution. This is also used in other delay measurement approaches, e.g., [5]–[7].

B. Methodology

For the time stamp compression part, we use the wavelet compression. There are other efficient wavelet compression methods [17]. The error of delays for different packets are averaged, leading to a small aggregated averaged delay in the order of 10^{-6} to 10^{-3}. Clearly, when the compression has a high distortion, the accuracy of delay measurement will be affected. However, the ordering and loss measurement will not be affected. Here, the compression method is not the focus of the evaluation. At the receiver side, we accordingly calculate per-packet delay by combining OPA and compressed time stamps. It should be noted that OPA needs the compressed time stamps while other approaches do not. We also demonstrate that our method can be used to detect abnormal delays that would otherwise be unable to reveal.

To compare the performance of OPA with existing approaches, we implement the methods of LDA [5], FineComb [7] and RLI [6]. Since RLI can provide per-packet delay measurement, we first compare the performance of OPA with RLI in terms of relative per-packet delay error. We also show that compared with RLI, OPA provides more accurate fine-grained delay measurements. Moreover, we show that OPA can also identify the delay anomalies.

Further, we compare OPA with LDA, FineComb, RLI in terms of

- average delay,
- standard deviation,
- overhead, and
- number of delays that can be calculated.

We also compare different approaches under different loss rate and reordering rate combinations.

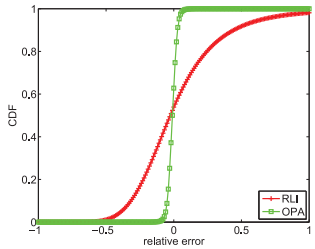


Fig. 8. Relative per-packet delay error of RLI and OPA.

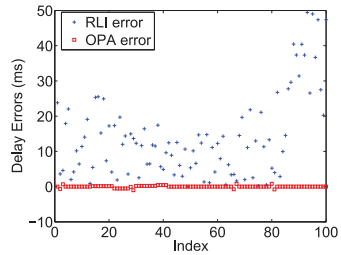


Fig. 9. The estimation error of large delays for RLI and OPA.

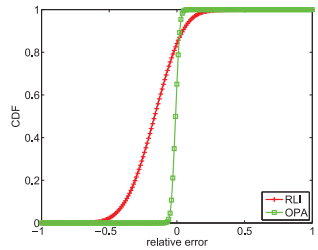


Fig. 10. Estimation error of large delays in RLI and OPA.

C. Fine-Grained Delay Measurement

We first compare OPA with RLI and evaluate the performance of OPA for fine-grained delay measurement. We vary the loss rate from 10^{-5} to a relatively high rate 10^{-3}. As we have introduced, OPA is effective to detect abnormal delays (e.g., large delays). We generate large delays with a probability of 10^{-3}. Meanwhile, we set p_b be the probability of a large delay given its previous packet has a large delay. Increasing p_b, the generated large delays will be more bursty. We set $p_b = 0.98$. Since the data rate in our evaluation is relatively stable, the reference packets are added in a fixed rate of $1/1000$. We also evaluate other rates for the reference packet, e.g., $1/300$ and $1/10$ as suggested in [6]. We find that the result is similar. Thus we choose a rate of $1/1000$ in order to reduce the overhead of RLI.

For comparison, we define the per-packet relative delay error and average delay error. Assume there are totally n packets and the true delay of packet i is $t_d(i)$ where $1 \leq i \leq n$. The average delay is calculated as $\bar{t}_d = \sum_{i=1}^{n} t_d(i)/n$. For an approach F, we denote the estimate delay for packet i by F with $t_d^F(i)$. Therefore, we calculate the per-packet relative error for packet i as

$$\epsilon_i^F = (t_d^F(i) - t_d(i))/t_d(i). \tag{10}$$

Meanwhile, we define the average delay error as

$$\eta = (\bar{t}_d^F - \bar{t}_d)/\bar{t}_d. \tag{11}$$

It should be noted that we can only calculate per-packet relative delay error for RLI and OPA. For other protocols, we cannot calculate the per-packet delay and the per-packet relative delay error. We compare the per-packet relative delay error for OPA and RLI. Fig. 8 shows the CDF of per-packet relative delay error. We can see that the relative per-packet delay error of RLI is larger than that of OPA. The relative per-packet delay error is very small for OPA.

To investigate how OPA can be used to calculate per-packet delay, we compare the delay calculated by RLI and OPA with the original delay. We check if RLI and OPA can correctly identify those large delays. Fig. 9 shows the error of those large delays. We can see that the error of RLI for most large delays is larger than that of OPA, because RLI uses interpolation based approach between reference packets. Since large delays are infrequent, packets with large delays may not be chosen as reference packets. Therefore, the delay calculated from interpolation cannot reflect those large delays, resulting in estimation error. This also explains Fig. 8 that RLI has a high estimation error. We can also see that OPA can effectively estimate each packet especially for those large delays. In Fig. 9, we only plot the errors for large delays for clarity.

To further investigate the performance, we calculate cumulative distribution of per-packet relative delay error for those large delays in Fig. 10. In Fig. 10, we show the cumulative distribution of delay error for RLI and OPA respectively. First, per-packet delay can be efficiently recovered with OPA since most of relative errors are distributed between -0.05 and 0.05. The relative per-packet delay error of RLI is larger than OPA. Second, those large delay are smoothed in RLI, leading to a negative relative error as shown in Fig. 10.

D. Comparison With Existing Approaches (LDA, FineComb, RLI)

In additional to per-packet delay, we also compare OPA with existing approaches (LDA, FineComb and RLI). We compare the overhead and ratio of recovered delays (i.e., delays that can be calculated) for those four approaches in order to examine their practical performance. The reordering rate is set to 10^{-5} in those evaluations.

Fig. 11(a) shows the comparison of average delay error. We can see that all those four approaches have a very small error for average delay. Due to a small reordering rate (10^{-5}) and the sampling, the difference between LDA and FineComb is very small.

Fig. 11(b) shows the comparison of error for standard deviation. We can see that all four approaches exhibit a low relative error. RLI has the largest relative error. As we have explained, RLI cannot identify large delays, leading to a large error in average delay and standard deviation. It should be noted that RLI is proposed to be applied to delays with locality. It is less beneficial to apply RLI to other delay distributions.

Fig. 11(c) shows the communication overhead from the sender to the receiver for delay calculation. RLI and OPA can calculate per-packet delay while LDA and FineComb can only calculate the aggregated delay statistics. The communication overhead of LDA is determined by the number of rows. The

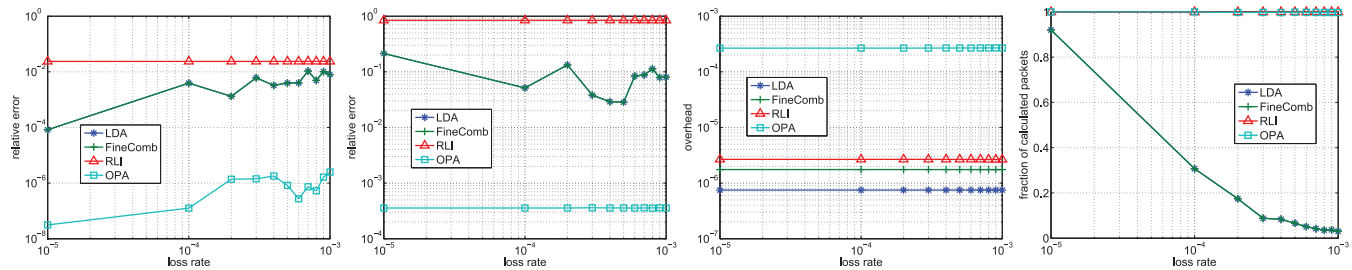


Fig. 11. Comparison of LDA, FineComb, RLI and OPA. (a) Error of average delay, (b) error of standard deviation, (c) communication overhead, and (d) fraction of delays that can be calculated.

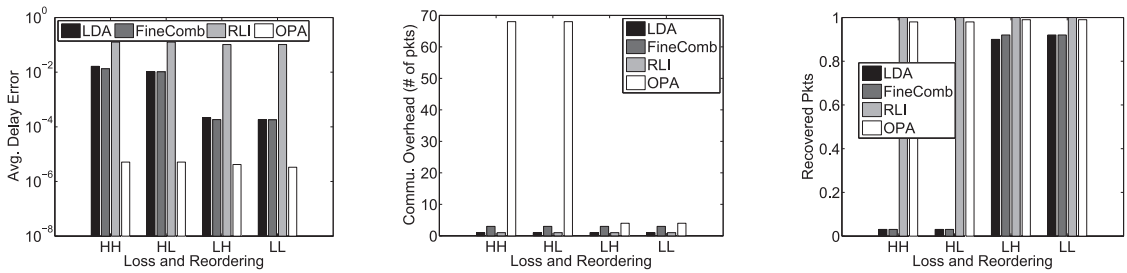


Fig. 12. Performance of different combinations for different approaches. (a) Average delay error. (b) Communication overhead. (c) Recovered packets.

communication overhead of FineComb is determined by the number of groups and the size of stash (100). The communication overhead of RLI depends on the number of reference packets. The communication overhead of OPA depends on the number of groups and the compression ratio. Fig. 11(c) shows that the overhead of OPA and RLI is higher than that of the other two approaches. Meanwhile, OPA and RLI provide per-packet measurement rather than aggregated delay statistics. The overhead of OPA is in the order of 10^{-4}, which indicates OPA only needs several packets to measure delay for 10000 packets. The result also coincides with the analytical result in Section VI. It can also be seen that RLI has a relative stable overhead while the error rate is increasing.

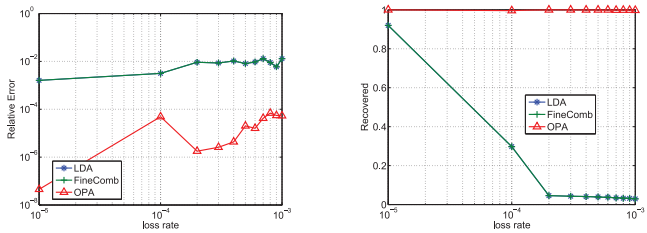
Fig. 11(d) shows the fraction of recovered delays (i.e., delays that can be calculated). In LDA and FineComb, a group is useless if the group contains lost packets. The packet losses are reduced after sampling. However, the total number of recovered delays is also reduced. In RLI, almost all delays can be calculated except for lost packets. For OPA, theoretically, OPA can recover all groups of packets with losses and reordering. Practically, we only calculate the delays in groups with no more than two errors. There may exist groups with more than two errors and thus those groups are not recovered. However, as long as the number of group is carefully chosen as we have shown, the number of such kind of groups should be very small. As shown in Fig. 11(d), the number of recovered delays is high for OPA and RLI, which is higher than that of the other two approaches.

E. Different Error Rate Combinations

We also evaluate the performance for different approaches under different loss rates and reordering rates. In this evaluation, we have relatively high loss/reordering rate (10^{-3}, denoted with H) and relatively low loss/reordering rate (10^{-5}, denoted with L). We use two characters to denote different loss and reordering rate combinations, and then examine the performance of different combinations. The first character indicates the loss rate and the second one indicates the reordering rate. For example, HL means a relatively high loss rate and a low reordering rate. Fig. 12 shows the results. We can see that OPA can achieve the smallest average delay error. Due to large delays, RLI has a larger delay error than other approaches. It should also be noted that RLI performs well for delays with locality. For communication overhead, OPA has the largest overhead while RLI has the smallest overhead since RLI only needs to send reference packets to the receiver. For the recovered packets, OPA can recover most of the packets. For RLI, it can always recover all packets since it uses interpolation based approach. In LDA and FineComb, groups with loss packets are discarded, leading to less recovered packets than OPA and RLI.

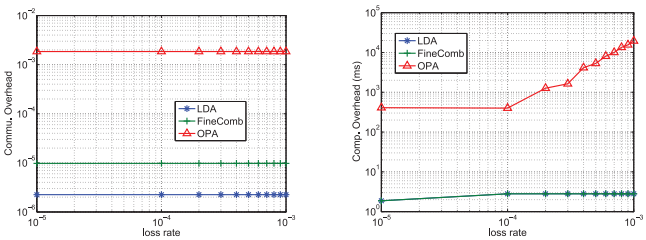
F. Handling Unknown Error Rate

We evaluate different approaches for handling unknown error rate. As we have introduced, LDA and FineComb use multi-bank approach to handle unknown error rate. We follow the configurations introduced in LDA [5]. More specifically, the sampling probability is set to $p = a \times m/(L+1)$, where $a = 0.5$, $m = 100$ is the number of rows (i.e., groups in each bank) and L is the number of packet losses. As in LDA, we set the probabilities to 0.00001, 0.0001, and 0.001. The probabilities can also be set to other values. Fig. 13(a) shows the result for relative error for different loss rates with multi-bank LDA and FineComb. For a loss rate in the interval between two probabilities (e.g., 0.00001 and 0.0001), the smaller the loss rate is, the higher accuracy OPA can achieve. Fig. 13(b) shows the recovered packets for different methods. OPA has a high recovered ratio. In OPA, we do not maintain disjoint set for different bank



(a) (b)

Fig. 13. Evaluation of multi-bank approach for unknown error rate. (a) The average delay error for different methods. (b) The calculated packets for different methods.



(a) (b)

Fig. 14. Overhead for unknown error rate. (a) The communication overhead for different methods. (b) The computation overhead for different methods.

and thus a packet can be used for different banks. Due to sampling, the effective sampling size of LDA and FineComb are decreasing as loss rate increasing. The effective sampling size of LDA and FineComb is calculated from the bank chosen for delay estimation as introduced in LDA.

Fig. 14(a) shows the overall communication overhead. It can be seen that the communication overhead is increased by using multiple banks compared with the overhead for a single bank. As increasing of loss rate, the computation overhead of OPA increases. This is because the when the number of errors increases, the average number of errors in each group for a certain bank will increase. This further leads to increase of the overall computation overhead. Meanwhile, the computation overhead of LDA and FineComb increases slightly.

VIII. Limitations and Future Work

While OPA can improve the accuracy of delay measurement by finding the loss and reordering, it incurs additional overhead compared with existing approaches such as LDA and FineComb. On the other hand, LDA and FineComb only need to calculate the sum of time stamps and the stash. The overhead is proportional to the number of packets. Thus the overhead of OPA is much higher than that of LDA and FineComb. In the analysis section, we have shown the communication overhead and the computation time. We can see that if the data rate is very high, e.g., 10 Gbps which is not uncommon, the overhead will be too high to run OPA online on the router. Therefore, it is better to use OPA for low data rate link or for a particular flow in a high data link. For example, a network manager may need to investigate the conditions of a particular flow. OPA can also be used for offline analysis to derive the packet delay. We hope that future research can further reduce the overhead and make the proposed method more practical and efficient in production networks.

IX. Conclusion

Fine-grained delay measurement is critical to understand and improve system performance. We design a new data structure named order preserving aggregator (OPA) to efficiently recover the ordering and identify the losses. OPA exploits intrinsic data properties, i.e., most packets are order-preserved and correct. Based on OPA, we propose a two-layer design for efficient per-packet delay measurement. We implement OPA and the evaluation results show that OPA achieves per-packet delay with 2% relative error while only incurring overhead in the order of 10^{-4} with respect to the number of data packets. We believe OPA can be widely used for per-packet delay measurement and ordering recovery in system management, performance monitoring and diagnosis. OPA also has some limitations regarding the computation and communication overhead. We also hope that future research can further reduce the overhead of OPA and make it more practical and efficient in production networks.

Appendix
Constructing the Coefficient Matrix

We require that any α columns in the submatrix $A_\sigma[1,\alpha]$ are independent. Here we leverage the Vandermonde matrix to construct the matrix A, i.e.,

$$A = \begin{pmatrix} 1 & 1 & \dots & 1 \\ a_1 & a_2 & \dots & a_w \\ a_1^2 & a_2^2 & \dots & a_w^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1^k & a_2^k & \dots & a_w^k \end{pmatrix} \tag{12}$$

where $a_i \neq a_j$ for $i \neq j$. We have the following theorem:

Theorem 1: Any α columns in submatrix $A_\sigma[1,\alpha]$ are independent.

Proof: Denote the matrix consisting of any α columns from matrix $A_\sigma[1,\alpha]$ as $A_\sigma^\alpha[1,\alpha]$. We have

$$A_\sigma^\alpha[1,\alpha] = \begin{pmatrix} 1 & 1 & \dots & 1 \\ a_{i_1} & a_{i_2} & \dots & a_{i_\alpha} \\ a_{i_1}^2 & a_{i_2}^2 & \dots & a_{i_\alpha}^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_1}^{\alpha-1} & a_{i_2}^{\alpha-1} & \dots & a_{i_\alpha}^{\alpha-1} \end{pmatrix} \tag{13}$$

where $1 \leq i_k \leq w$ for $1 \leq k \leq \alpha$. The determinant of matrix $A_\sigma^\alpha[1,\alpha]$ can be calculated as

$$det(A_\sigma^\alpha[1,\alpha]) = \prod_{1 \leq k < j \leq \alpha} (a_{i_k} - a_{i_j}). \tag{14}$$

As $a_{i_k} \neq a_{i_j}$ for all $k \neq j$, we can see that $det(A_\sigma^\alpha[1,\alpha]) \neq 0$. Thus any α columns from matrix $A_\sigma[1,\alpha]$ are independent. □

Normally, we only need one checking row (checking equation) for each group. Therefore, for a group with α lost packets, we require that matrix A has $\alpha + 1$ rows. Then we can use the first α rows for the mapping equation and the last row (row $\alpha + 1$) for the checking equation.

REFERENCES

[1] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, design and analysis of a large-scale P-VOD system," in *Proc. SIGCOMM*, 2008, p. 2.

[2] M. Keller, J. Beutel, and L. Thiele, "How was your journey? Uncovering routing dynamics in deployed sensor networks with multi-hop network tomography," in *Proc. ACM SenSys*, 2012, pp. 15–28.

[3] R. Martin, "Wall Street's quest to process data at the speed of light," *InformationWeek* 2007 [Online]. Available: http://www.informationweek.com/news/infrastructure/showArticle.jhtml?articleID=199200297

[4] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot, "Measurement and analysis of single-hop delay on an IP backbone network," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 6, pp. 908–921, Aug. 2003.

[5] R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese, "Every microsecond counts: Tracking fine-grain latencies with a lossy difference aggregator," in *Proc. ACM SIGCOMM*, 2009, pp. 255–266.

[6] M. Lee, N. G. Duffield, and R. R. Kompella, "Not all microseconds are equal: Fine-grained per-flow measurements with reference latency interpolation," in *Proc. ACM SIGCOMM*, 2010, pp. 27–38.

[7] M. Lee, S. Goldberg, R. R. Kompella, and G. Varghese, "Fine-grained latency and loss measurements in the presence of reordering," in *Proc. SIGMETRICS*, 2011, pp. 329–340.

[8] M. Keller, L. Thiele, and J. Beutel, "Reconstruction of the correct temporal order of sensor network data," in *Proc. IPSN*, 2011, pp. 282–293.

[9] M. Shahzad and A. X. Liu, "Noise can help: Accurate and efficient per-flow latency measurement without packet probing and time stamping," in *Proc. ACM SIGMETRICS*, 2014, pp. 207–219.

[10] M. Lee, N. Duffield, and R. R. Kompella, "MAPLE: A scalable architecture for maintaining packet latency measurements," in *Proc. ACM SIGCOMM IMC*, 2012, pp. 101–114.

[11] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proc. ACM SIGCOMM*, 2012, pp. 115–126.

[12] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM*, 2011, pp. 50–61.

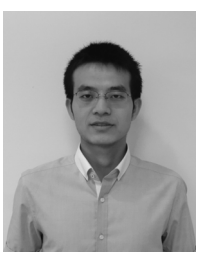
[13] L. Chen, B. Li, and B. Li, "On meeting deadlines in datacenter networks," *Tsinghua Sci. Technol.*, vol. 18, no. 3, pp. 273–285, 2013.

[14] Y. Gao, W. Dong, C. Chen, J. Bu, and X. Liu, "Towards reconstructing routing paths in large scale sensor networks," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 281–293, Jan/ 2016.

[15] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE/ANSI 1588 Standard, 2002.

[16] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proc. USENIX OSDI*, 2002, pp. 147–163.

[17] O. Rioul and P. Duhamel, "Fast algorithms for discrete and continuous wavelet transforms," *IEEE Trans. Inf. Theory*, vol. 38, no. 2, pp. 569–586, Mar. 1992.



Jiliang Wang is currently an Assistant Professor with the School of Software, Tsinghua University, Beijing, China. He received the B.E. degree from the University of Science and Technology of China, Hefei, China, and the Ph.D. degree from Hong Kong University of Science and Technology, Hong Kong. His research interests include wireless and sensor networks, mobile network, and pervasive computing. He is a member of the IEEE and the ACM.

Shuo Lian received the Ph.D. degree from the School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China. His research interests include wireless and sensor networks, network measurement, and pervasive computing.

Wei Dong (S'08–M'11) received the B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 2005 and 2011, respectively. He is currently an Associate Professor with the College of Computer Science, Zhejiang University. His research interests include network measurement, wireless and mobile computing, and sensor networks. He is a member of the IEEE and the ACM.

Xiang-Yang Li (SM'08–F'15) is a Professor and Executive Dean with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China, and was a Professor with the Illinois Institute of Technology, Chicago, IL, USA. He is an IEEE Fellow, ACM Distinguished Scientist since 2015, and holds the EMC-Endowed Visiting Chair Professorship at Tsinghua University from 2014 to 2016. He is a recipient of the China NSF Outstanding Overseas Young Researcher (B). He received the M.S. (2000) and PhD (2001) degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2000 and 2001, respectively, and the Bachelor degree from the Department of Computer Science and the Bachelor degree from the Department of Business Management, Tsinghua University, Beijing, China, both in 1995. His research interests include wireless networking, mobile computing, security and privacy, cyber physical systems, and algorithms. He has won six Best Paper awards, and one Best Demo award. He published a monograph "Wireless Ad Hoc and Sensor Networks: Theory and Applications" and co-edited several books, including *Encyclopedia of Algorithms*. Dr. Li is an Editor of several journals and has served many international conferences in various capacities

Yunhao Liu received the B.S. degree from the Automation Department, Tsinghua University, Beijing, China, and the M.A. degree from Beijing Foreign Studies University, China. He received the M.S. and Ph.D. degrees in computer science and engineering from Michigan State University, East Lansing, MI, USA. He is now the Dean of the School of Software and holds the ChangJiang Chair Professorship with Tsinghua University. He is a Fellow of the IEEE and the ACM. His research interests include sensor network and IoT, localization, RFID, distributed systems, and cloud computing.