Every Packet Counts: Fine-Grained Delay and Loss Measurement with Reordering

Jiliang Wang[1], Shuo Lian[2], Wei Dong[3], Yunhao Liu[1] and Xiang-Yang Li[4]
1. School of Software and TNLIST, Tsinghua University
2. School of Electronic and Information Engineering, Xi'an Jiaotong University
3. College of Computer Science, Zhejiang University
4. Department of Computer Science, Illinois Institute of Technology
{jiliang, yunhao}@greenorbs.com, lianshuo@mail.xjtu.edu.cn, dongw@zju.edu.cn, xli@cs.iit.edu

***Abstract*—Delay is an important metric to understand and improve system performance. While existing approaches focus on aggregate delay statistics in pre-programmed granularity, providing only statistical results such as averages and deviations, those approaches fail to provide fine-grained delay measurement at a flexible level and thus may miss important delay characteristics. For example, delay anomalies, which are critical system performance indicators, may not be captured by existing coarse-grained approaches. In this work, we propose a fine-grained delay measurement approach based on a new measurement structure design called order preserving aggregator (OPA). OPA can efficiently encode the ordering and loss information by exploiting inherent data characteristics. Based on OPA, we propose a two-layer design to convey both ordering and time stamp information, and then derive per-packet delay/loss measurement with a small overhead. We evaluate our approach both analytically and experimentally with widely used real-world data sets. The results show that our approach can achieve accurate 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 less than 4×10^{-4} additional overhead.**

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 will significantly impact the quality of service for applications such as voice-over-IP, which would further impact the user experience. Moreover, for critical automatic trading system with a stringent delay requirement, 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 protocol designers and system users care about. It is of great importance to understand and improve system performance. Consequently, delay performance measurements have attracted a lot of research efforts [4] [5] [6] [7] [2] [8] [9].

Intuitively, delay can be measured by embedding a time stamp to record the sending time in each packet at the sender. When a packet is received, the receiving time is recorded and 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 Internet routers, it is difficult to modify the IP packets. Even if a time stamp can be inserted, this intrusive behavior may result in non-negligible overhead or unforeseen protocol behaviors [5].

/978-1-4799-6204-4/14$31.00 ©2014 IEEE

To satisfy practical system requirements, a common applicable approach is to measure packet delay non-intrusively without modifications to data packets. Following such a design principle, different methods, e.g., LDA, FineComb, are proposed [5] [6] [7] for providing aggregate 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

While existing approaches focus on providing aggregated delay statistics at a pre-programmed granularity, they cannot provide flexible fine-grained delay measurements. On the other hand, as revealed in existing works [10], fine-grained delay measurements are of great importance in performance monitoring, system diagnosis, traffic engineering and etc.

First, fine-grained delay measurement is required to reveal 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 clearly different from the case that all 1000 packets have a delay of 2 ms, though both cases exhibit the same aggregated average delay. Moreover, even for the same average and deviation, different types of packets may exhibit 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 whole set of packets.

Second, fine-grained delay performance is critical to network diagnosis [10]. In diagnosis it is important to investigate a special type of packets, e.g., DNS packet, ACK packet. Existing approaches cannot provide delay measurement at such a granularity. Moreover, with fine-grained delay measurements delay anomalies [11] [12] [13] can be revealed. For example, in a time-critical bidding system, the bidding request packet

usually has a tight deadline [10]. Failure to meet the deadline will miss some bidding opportunity or even result in significant profit loss. Thus measuring the per-packet delay of the request packet should be important to system diagnosis in real time bidding system, high performance computing data center and etc.

Third, fine-grained delay measurement is required or presumed in various protocols. It can be leveraged to significantly improve the system performance. For example, many protocols in Internet and data center [14] [15] show that incorporating fine-grained delay measurement would improve the system performance. In [2], it has also shown that fine-grained delay measurements can be used to recover routing dynamics and improve routing performance.

Last but not least, inherently aggregated delay performance cannot be accurately calculated in presence of packet losses or reorderings [5] [6] [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. On the other hand, from the system management perspective, groups of packets with losses or reorderings should be particularly important to understand system behavior and reliability. Therefore, accurate delay measurements for such groups should be very important and provide useful information that is otherwise difficult to obtain.

C. Our approach

To address those problems, we need a flexible fine-grained delay measurement approach to fill the gap. We propose a fine-grained delay and loss measurement approach. We design a new data structure called order preserving aggregator (OPA), with which packet loss as well as ordering information can be efficiently represented and recovered with a small overhead. The OPA design exploits the intrinsic data properties that lost and reordering packets should usually be much less than legitimate packets, allowing efficient ordering and loss information representation and recovery. Compared with existing works, the OPA design incurs an overhead related to the number of lost and reordering packets rather than a fixed sampling/probing rate or the total number of packets in existing works [6].

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

Compared with existing approaches, our approach has several merits. First, our approach provides per-packet delay and loss measurement, while existing approaches only provide aggregated statistics. Second, our approach explores the inherent data properties, and thus incurs a low computation and communication overhead. Third, packet delays in groups with losses or reorderings, which should be of great importance but cannot be measured in traditional approaches, can be calculated with our approach. While our approach is proposed to measure delay in Internet and data center, it can also be extended for other networks such as wireless ad hoc networks [2], in which computation and energy resources are very limited and efficient delay measurement is very crucial.

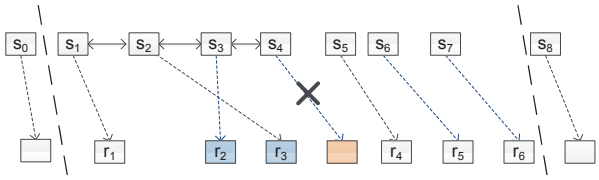

Fig. 1: Fine-grained delay measurement with packet loss and reordering.

The contributions are summarized as follows.

- *Architecture for fine-grained delay measurement.* We propose the OPA design, a new measurement structure to efficiently represent and recover the ordering and loss information by exploiting data properties. Based on OPA, we design a two-layered information representation system for efficient fine-grained delay measurement with a small overhead.
- *Performance analysis.* We analyze the correctness of our approach. Further, we also analyze the computation and communication overhead for the proposed approach.
- *Evaluation with real data sets.* We evaluate our approach with real-world data sets. The results demonstrate the effectiveness of our approach. With only 4 additional packets for every 10^4 data packets, which is even less than traditional probing based methods (e.g., about 10000 per second [5]), per-packet delay can be measured with an average relative error at 2%, and the aggregated delay can be masured 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 used in our measurement. Section III introduces existing approaches for delay measurement. Section IV introduces the framework overview of the delay measurement method and the design of OPA. Section V shows how to transmit time stamp information and how to leverage OPA for per-packet delay measurement. Section VI shows the analytical results of the proposed method. Section VII shows the evaluation results of the proposed approach and Section VIII concludes this work.

II. Assumptions and Network Model

We consider measuring packet delay from one router to another router. We denote the router from which the packets are transmitted as $sender$ and the other as $receiver$. On the path from the sender to the receiver, there may be multiple intermediate routers. Our method can also be applied to delay measurement between the ingress and egress points on the same router as in [5] or two nodes in wireless ad hoc networks [2]. We divide a sequence of packets into segments and measure packet delay in each segment. 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 Figure 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, e.g., 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, packets between the same delimiter packets at the sender are not necessarily the same with those at the receiver. The ordering of packets for segments between the same delimiter packets at the sender and receiver are also not necessarily the same. For example, as shown in Figure 1, packet s_2 is a reordering packet and packet s_4 is a lost packet. The packets orderings at the sender and receiver, for segments between the same delimiter packets, are no longer the same. It should be noted that delimiter packets may also be lost. If a delimiter packet is lost, we discard the corresponding segment and move to the next segment. Here we assume delimiter packets are successfully received.

As in previous works [5] [6] [7], we assume that the sender and receiver are synchronized. This can be accomplished by existing time synchronization protocols [16] [17]. As in real networks, there is no common sequence number in packets since packets may come from different sources with different protocols [5] [7]. Meanwhile, the packet cannot 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 record 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 stamp subtracted by the sending time stamp. For example, if packet r_i corresponds to s_j at the sender, the delay can be calculated as $t_d(i) = t_i^r - t_j^s$. Hereafter, we focus our description on how to measure per-packet delay for a segment with n packets from the sender.

III. Existing Approaches

A. Timestamping based approaches

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

B. Probing based approaches

Probing is a commonly used technique to estimate the 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 the same behavior with other packets. Then by measuring the statistics of the probing packets, the statistics for other packets can also be estimated. Probing based methods, which significantly reduce the measurement overhead, fail to achieve fine-grained delay measurement results [5] [6] [7].

C. LDA

Loss Difference Aggregator (LDA) is proposed to obtain the estimation of delay averages and deviations with a small overhead. In LDA, packets are also divided into groups by a certain hash function. For each group of packets, the sum of the time stamps and the number of packets are sent to the receiver. Upon receiving such information, the receiver first find the corresponding group of packets by applying the same hash function. If the total packet counter for the group from the sender matches that at the receiver, the receiver treats the two groups as the same. Otherwise the receiver will discard the entire group. Then the receiver can calculate the average delay for such a group as follows. For each group, the sum of delays can be calculated by substracting the sum of the 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 number of packets. Therefore, LDA significantly reduces the measurement overhead by only transmitting the sum of time stamps and packet counter. However, when there exist packet loss or reordering in a group, the entire group has to be discarded and the delay for packets in such a group cannot be measured. With even a single loss or reordering packet, the entire group becomes useless. While LDA provides aggregate delay results, it cannot achieve fine-grained per-packet delays.

D. FineComb

In LDA, packets that belong to one segment may be misidentified into other segments due to packet reordering. While dividing a segment of packets into groups, some groups may have packets from other segments or miss some packets. In such a case, those groups of packets cannot be used for delay measurement. To address this problem, FineComb [7] uses a special data structure called stash to recover the correct packets for a segment. Stash is a structure to maintain the information of packets near the boundary of segments. FineComb uses an exhaustive search to check whether a packet in the stash belongs to a particular group. FineComb needs to enumerate all possible combinations for packets in the stash, which introduces a high overhead. Moreover, for groups with lost packets, FineComb cannot calculate the delay for those groups.

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 the reference packets, RLI uses interpolation to estimate delays for packets between the reference packets. With such a method, per-packet delay can be estimated. However, the interpolation based method inherently assumes a specific delay distribution (e.g., linear delay distribution) for reference packets and packets inbetween.

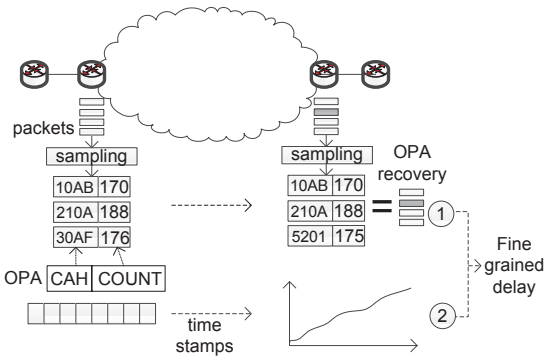

Fig. 2: The design overview of OPA for fine-grained delay and loss measurement.

Thus it may not be able to accurately measure the per-packet delay in presence of frequent delay changes or significant delay variations. For example, a sudden increase of packet delays between two reference packets cannot be captured, resulting in missing of important information to investigate variations or anomalies.

F. MAPLE

Recently, MAPLE [10] is proposed to store and query per-packet delay. MAPLE uses a scalable data structure to efficiently store and query fine-grained delay information. MAPLE can be leveraged as the storage and query system for fine-grained delay in our system. MAPLE focuses on delay storage and query rather than per-packet delay measurement.

IV. OPA Design

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

- The method should be non-intrusive and should not require any modification to data packets. Thus the method does not introduce any change to existing routing protocols.
- The method should be light-weight and efficient, without incurring much additional computation and communication overhead. This is very important for practical use of the method.
- The abnormal delay, e.g., large delays, should be captured and preserved in order to investigate the important system metrics.

A. Design framework

A straightforward approach is to send the time stamp information from the sender to the receiver. More specifically, we attach a packet ID to each time stamp in order to calculate the per-packet delay at the receiver side. This clearly introduces a high overhead. Moreover, an immediate improved approach 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 presence of packet loss and reordering. For a single packet loss or reordering, the compressed information cannot be used at the receiver since the receiver does not know which one is lost. For example, as shown in Figure 1, assume all packets are correctly received except that packet s_4 is lost. 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). Consequently the delay is calculated as $t_5^r - t_5^s$, which is actually incorrect for packet r_5. 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 the per-packet delay. This problem is further exacerbated when there exists packet reordering.

Our basic idea is to efficiently transmit the ordering and time stamp information from the sender to the receiver. The ordering information contains the position for each packet from the sender. At the receiver side, not only the time stamps should be recovered but also the correct ordering and lossy information in order to calculate the per-packet delay.

It is challenging to represent the ordering and time stamp information due to reasons from two aspects. First, there is a significant difference between ordering information and time stamp information. The ordering information should be exactly recovered while the time stamp information can have a small relative error. Second, even for the ordering information, it is difficult to represent due to a large number of possible orderings. For example, the possible ordering for a group of n packets is $n!$. Considering for possible losses, the number of combinations is even much higher. Therefore, using lossless compression, which is a simple and straightforward idea for the ordering information, leads to a low compression ratio and thus is not suitable for ordering information. On the other hand, lossy compression, which recovers the time stamp information and has a high compression ratio, cannot recover the ordering information and hence cannot be used for per-packet delay calculation. This introduces challenges to transmit and recover the ordering information at the receiver.

As shown in Figure 2, we design a two-layer representation system. There are several properties that motivate our design. First, in practical networks the reordering as well as loss event will not be frequent. Normally, the ordering for received packets should be very similar to that for packets at the sender. Intuitively, based on the received information, the ordering "difference" between the sender and receiver should be small. Thus instead of transmitting all possible orderings, the information that needs to be conveyed can be significantly reduced considering the small ordering difference. Second, the sending time stamps for packets sent sequentially should be easy to compress.

More specifically, packets are first divided into groups at the sender side. For each group, the sender calculates the corresponding OPA (the first layer) as well as the compressed time stamps (the second layer), and then transmit the two-layer information to the receiver. At the receiver, the receiver can efficiently recover packet ordering and lost packets 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

At the sender side, we first divide the packets of each segment into groups based on some 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., $n = g \times w$. Later, we deal with the case that the number of packets in different groups are not equal in Section V. Using a randomized hash function can also divide bursty packet losses (e.g., due to congestion or buffer overflow) into different groups.

To simplify the presentation, we annotate an unique ascending sequence number to each packet in a group according to its sending order. Hence, 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.

We leverage the hash value to identify each packet. Since the hash value does not contain any ordering information, we use a subtle method to encode the ordering information in the hash value. For each group G_i, the sender calculates the augmented hash value of the jth packet s_{ij}, i.e., $h_{ij} = Hash(s_{ij}{+}{+}j)$, where $++$ is the string concatenation operation. We call h_{ij} as the augmented hash of the jth packet in group G_i. We also 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 a group G_i, the sender calculates the augmented hash values for all packets. At the receiver side, if the packet ordering is exactly the same with that at the sender, the augmented hash values should be exactly the same. Thus the receiver can check the ordering information by transmitting the augmented hash values from the sender to the receiver. However, this would still incur a high overhead as there is an augmented hash value for each packet. Meanwhile, this cannot be used for the case with lost packets. As we have mentioned, comparing to the total number of packets in each group, the reordering and lost packets would be very rare. Thus, the received ordering for each group of packets, which is similar to the ordering for packets from the sender, contains useful information that should be exploited.

In our approach we do not send augmented hash values of all packets to the receiver. Intuitively, we only send the reordering information to the receiver. To achieve such a goal, we send the linear combinations of the augmented hash values for all packets in the group. More specifically, the sender calculates k linear combinations $B_i = (b_{i1}, b_{i2}, \ldots, b_{ik})^T$ for w packets in group G_i 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. Later we will introduce how to determine the coefficient matrix. 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. Eq. 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 the H_i has w values. For each 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). For each segment, the sender can calculate the OPAs for all groups in order. Then the sender can aggregate tuples for multiple groups (e.g., in our implementation, the combined augmented hash has a length of 24 bytes and w has a length of 4 bytes) 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.

First, the receiver uses the same delimiter packets from the sender to find the corresponding segment.

Second, the receiver divides packets of a segment into groups using the same method (i.e., $Hash(\cdot)$) with the sender. Assume the received groups are $G'_1, G'_2, \ldots, G'_g$. For the received group G'_i, we denote those packets as $G'_i = (r_{i1}, r_{i2}, \ldots, r_{iw'})$. We also denote the augmented hash for a received packet r_{ij} as $h'_{ij} = Hash(r_{ij}{+}{+}j)$ and accordingly $H'_i = (h'_{i1}, h'_{i2}, \ldots, h'_{iw'})^T$. For group G_i, the received OPA_i = (the Order preserving aggregator B_i, the number of packets w). Based on the received information, there are two different cases for a group G'_i.

- If $w' = w$, we then calculate $B'_i = AH'_i$. We require that the first k rows in matrix A are independent. If $B_i = B'_i$, the group of packets in G'_i should be same with the packets in G_i from the sender. Notice that if $G'_i \neq G_i$, the probability for $B_i = B'_i$, i.e., $AH'_i = AH_i$ should be very low. Meanwhile, since we use the augmented hash for each packet in the group, the ordering of packets at the receiver should also be same with that at the sender. Based on the OPAs, we can check if the packet ordering are persevered at the receiver, and we can also efficiently check if the two groups are exactly the same.
- $w' \neq w$ or $B'_i \neq B_i$, those two groups of packets are not the same. This means that there may be packet losses or packet reorderings from group G_i to G'_i.

D. OPA recovery

We now deal with the case when there are packet losses or packet reorderings in a group. First, we consider the case of 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 lost packets and reordering packets. When the number

of lost packets α is less than k, we show that the order preserving aggregator can be used to derive the reordering packets, the lost packets and their corresponding positions. Later we explain how to deal with the case when the number of packet loss α is unknown.

Even we know there are $\alpha(\alpha < k)$ packet losses in a group, it is difficult to derive information for lost packets. First, we do not know the correct position information for the w' received packets in this group and we have no information about which α packets are lost. Second, due to packet losses, the receiver does not have enough augmented hash values to calculate CAH B'_i. It is difficult to use the CAH information. For the lost packets, by no means we can calculate the delay for those packets. Here, our goal is to determine the positions of the lost packets and the positions of the received w' packets.

Therefore, we need to determine the positions for the received w' packets among the w packets from the sender, i.e., mapping the receiving w' packets to the correct positions from the sender. 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 G'_i is the same with 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. Generally, 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 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 can obtain a new ordering and position of packets. For example, if the vector (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 the vector $v = (v_1, v_2, v_3)$ 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 Eq. 2, we have

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

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 of lost packets. The sum of those two parts should be the CAHs.

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 $\bar{\sigma} = \{i |\ \nexists_j \sigma_j = i$ for all j and $1 \leq i \leq w\}$. 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 $|\bar{\sigma}|$. Eq. (3) can be written as

$$A_\sigma \times H_\sigma + A_{\bar{\sigma}} \times H_{\bar{\sigma}} = B_i. \tag{4}$$

There are two challenges for us to calculate the position information for lost packets and received packets according to the equation. First, we do not know the position of the received packets and thus we do not have information for H_σ. Second, we do not have the information for the lost packets. Thus we do not have information of $H_{\bar{\sigma}}$. Thus both H_σ and $H_{\bar{\sigma}}$ are unknown in Eq. 4. Meanwhile, we do no know the mapping vector σ of the received packets, generally the equation cannot be solved.

While we cannot solve the equation to obtain the value of H_σ and $H_{\bar{\sigma}}$, here we design a subtle method. In this method, we leverage the properties of packets in G_i and G'_i, to first calculate the value of σ without solving the equation.

We first consider different cases for Eq. 4. First, if there are no lost and reordering packets, we have $w' = w$ and $\sigma = (1, 2, \ldots, w)$. In this case we can easily obtain H_σ and $H_{\bar{\sigma}}$. This case can be easily verified with Eq. 4. Second, there are some packet losses. In most cases, the lost packets should be much less comparing to the number of received packets. Thus we can enumerate all possible packet losses and reorderings. For each possible packet loss and reordering, we can obtain a mapping σ. Based on σ, we can solve Eq. 4. Intuitively, as long as the number of lost packets and reordering packets is much less than the number of normal packets, the Eq. 4 can be solved.

Without loss of generality, assume there are α packet losses. Thus we have $|\bar{\sigma}| = \alpha$. Meanwhile, A_σ has $k - \alpha$ columns. For any mapping vector σ, we can calculate H_σ based on the received packets. Accordingly, we have the following *mapping equation*:

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

where $A_{\bar{\sigma}}[1, \alpha]$ is row 1 to row α of $A_{\bar{\sigma}}$, and similarly defined for $H_{\bar{\sigma}}[1, \alpha]$, $B_i[1, \alpha]$, $A_\sigma[1, \alpha]$ and $H_\sigma[1, \alpha]$. If we require that $A_{\bar{\sigma}}[1, \alpha]$ is invertible, the columns of $A_{\bar{\sigma}}[1, \alpha]$ are independent. Thus we can solve the above equation and obtain the value for $H_{\bar{\sigma}}$. After we have solved the Eq. 4, 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, to obtain the correct position information for the received and lost packets, we have the following two steps. First, the Eq. 4 should be solvable, i.e., the columns in $A^\alpha_{\bar{\sigma}}$ should be independent. Second, we should leverage the matrix A to verify the feasibility of the solution of $H_{\bar{\sigma}}$.

For the first step, we carefully construct the coefficient matrix A such that any α columns from the first α rows are independent. We will introduce more details about this step in the next subsection. In the second step, we require $\alpha < k$. As $\alpha < k$, we can use the remaining $k - \alpha$ rows as checking rows. Accordingly, we have the *checking equation*:

$$A_{\bar{\sigma}}[\alpha+1, k] \times H_{\bar{\sigma}}[\alpha+1, k] = B_i[\alpha+1, k] - A_\sigma[\alpha+1, k] \times H_\sigma[\alpha+1, k]. \tag{6}$$

In summary, here we have the mapping equation and checking equation. The mapping equation is used to find the possible mappings from the received packets to the sent packets. The checking equation is used to verify the feasibility of the

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 equation Eq. 5;
6: **if** The checking equation Eq. 6 is satisfied **then**
7: **return** $(\sigma, \bar{\sigma})$;
8: **return** (NULL, NULL);

mapping. If σ is a correct mapping vector, it should satisfy the checking equation. Otherwise, if σ is not a correct mapping vector, the checking equation cannot be satisfied.

Therefore, we can enumerate possible σ to calculate H_σ. Based on H_σ and the mapping equation Eq. 5, we can obtain $H_{\bar{\sigma}}$. Then we can check $H_{\bar{\sigma}}$ with the checking equation Eq. 6. Intuitively, the basic idea of the method is as follows. First, we enumerate all possible positions for the received packets. Based on the positions, we can obtain H_σ and $H_{\bar{\sigma}}$. Then we can further use the checking equation to check whether the solved H_σ and $H_{\bar{\sigma}}$ are feasible. If yes, we call σ a feasible mapping vector. Accordingly, we obtain the correct position for the received packets as well as the lost packets. Otherwise, the position vector σ is not correct. Notice that, as we will show later, considering the small number of packet losses and reorderings, we can significantly reduce the enumeration cost.

E. Constructing the coefficient matrix

To ensure that Eq. 5 is solvable, 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.,

$$A = \begin{pmatrix} 1 & 1 & \dots & 1 \\ a_1 & a_2 & \dots & a_w \\ a_1^2 & a_2^2 & \dots & a_w^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1^k & a_2^k & \dots & a_w^k \end{pmatrix} \tag{7}$$

where $a_i \neq a_j$ for $i \neq j$. It can be proved that any α columns in submatrix $A_\sigma[1, \alpha]$ are independent. This can be derived from the property of the matrix A. Therefore, we can always solve the mapping equation Eq. 5.

Normally, we only need one checking row (checking equation) for each group. Therefore, in a group with α lost packets, i.e., $|\bar{\sigma}| = \alpha$, we only require that the matrix A has $\alpha + 1$ rows. Then we can use the first α rows for mapping equation and the last row (row $\alpha + 1$) for the checking equation to verify if a solution is feasible. The basic algorithm for OPA recovery is shown in Algorithm 1. Line 2 means to recover the 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 $H_{\bar{\sigma}}$. As we can see, with more lost packets, the computation overhead is higher. For a group of w packets with α lost packets, the number of cases we need to check is $C(w, \alpha)$, i.e., the number of loops executed in Algorithm 1. Later, in Section VI, we will show how to reduce the computation overhead especially when the loss ratio for a segment is high. The proposed method only requires to send B_i of length $\alpha+1$ while the original method requires sending all the w time stamps and the packet IDs. Considering that in most cases $\alpha \ll w$, the proposed method significantly reduces the overhead.

Example. We use a simplified example to explain the basic procedure of our approach. Assume a group G_1 from the sender 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 augmented hash values for those three packets are $h_{11} = s_{11}{+}{+}1 = 21$, $h_{12} = s_{12}{+}{+}2 = 52$, and $h_{13} = s_{13}{+}{+}3 = 73$ and $H_1 = (h_{11}, h_{12}, h_{13})^T = (21, 52, 73)^T$, respectively;
- assume the coefficient matrix $A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$. Thus the CAH is calculated 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 packet counter w is 3;
- the OPA = ((146, 344), 3).

The receiver receives OPA = ((146, 344), 3). If the receiver packets are $G_1' = (2, 5, 7)$. We can calculate $H_1' = (21, 52, 73)$. Thus we have $B_1' = AH_1' = (146, 344)$ and $w' = 3$. Therefore, OPA′ = ((146, 344), 3) = OPA. Thus we can see from the received OPA that there is no loss and reordering. Then we can calculate the delay for each packet according to the compressed time stamps.

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, we calculate the number of lost packets as $\alpha = w' - w = 1$. Then according to Algorithm 1, we check the position of the lost packet. Assume that the first packet is lost, i.e., $(x, 2, 7)$, 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 $1 \cdot h_{11}' + 1 \cdot h_{12}' + 1 \cdot h_{13}' = 146$, we have $h_{11}' = 51$. Using this for the checking equation, we have $1 \cdot h_{11}' + 2 \cdot h_{12}' + 3 \cdot h_{13}' = 314 \neq 344$. 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 and obtain $\sigma = (1, 3)$.

V. Delay Calculation

After we have recovered the positions for all received packets, we then calculate the per-packet delay.

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: $t_d(i) \leftarrow t_i^r - \tilde{t}_{\sigma(i)}^s$;

A. Time stamp compression

To calculate the fine-grained delay, the sender needs to send the time stamp information to the receiver. With OPA, we can efficiently recover the ordering and loss information. We send the compressed time stamps to the receiver in the second layer. There are two properties for time stamps that facilitate the compression. First, the variation of each time stamp will not be large since packets are sent on a high speed data link, e.g., for a link up to several Gbps. Second, 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. In our implementation, we use wavelet for time stamp compression.

We have also observed two merits of the wavelet compression: the sum of error for all recovered time stamps is nearly zero-sum. Thus the average delay calculated is very closed to the true average. Second, the error is evenly distributed and very small, thus bursty and abnormal delays can be captured even in presence of time stamp error. It is worth noting that time stamp compression is not the focus of our method and other time stamp compression methods can be leveraged, e.g., [18] [19].

The receiver first recovers the sending time stamps. 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 position for the received packets. Thus we can calculate the delay for those w' packets by the receiving time stamp $-$ the sending time stamp, i.e., $t_d(i) = t_i^r - \tilde{t}_{\sigma(i)}^s$, where $\sigma(i)$ is the ith element in σ and $\sigma(i) \neq 0$. The main steps for calculate delay are shown in Algorithm 2. Line 2 and Line 2 show how 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

It can be seen that the reordering can also be processed with our proposed method. In presence of reordering, we need to accordingly enumerate possible positions and 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 position difference between the original position and the reordered position is often bounded by a shift d [10]. Thus in the mapping vector σ the possible positions are $x - d, x - d + 1, \ldots, x, x + 1, x + d$. This significantly reduces the computation overhead for solving the mapping equation. With a grouping method, the possible positions for a reordering packet are further reduced. For example, if we have 10 groups of packets, the number of possible positions for a packet in the grouping results is reduced from d to $d/10$. For reordering with no shift bound, the position cannot be identified with OPA.

C. 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 according to the construction method. As long as matrix A has more than w columns, the proposed method can still be used to recover the correct positions.

To achieve fine-grained delay measurement, it is required to maintain a buffer for all receiving time stamps. As our approach processes packets segment by segment, the maximum buffer size we need to maintain is the same with the segment size. In order to further reduce the memory overhead, we use the following approaches. First, we can choose an appropriate segment size to fit the memory constraint. Second, our approach can also be used in a on-demand manner. An user can specify particular constraints on the packets that need to be measured. For example, an user may be interested in packet delay for a particular protocol [12]. The user can measure packets in a particular flow or measure packets in a particular time period. In such cases, very limited information needs to be maintained by the router with our method. Recently, a fine-grained delay storage and query architecture is proposed [10]. This can be further combined with our approach for efficient per-packet storage and query.

It should also be noted that considering the low communication overhead and concise representation of OPA approach, the received information can be stored on the receiver. The stored information can then be used 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. In such a case, our method uses a random hash function to divide packets into groups. Bursty packets will be grouped into different groups. Thus the distribution of packet losses will impact the effectiveness of our approach.

VI. Analysis

A. Computation overhead

As introduced in Section IV, an important factor impacts the performance of our approach is the computation overhead. We need to enumerate different combinations of errors. In our approach, the computation overhead is significantly reduced after packet grouping considering that the loss rate is usually low. As we can see from the Algorithm 1 and Algorithm 2, the overhead is determined by the number of errors and the number of groups rather than a fixed probing rate. On one hand, with more groups, the number of OPAs as well as the

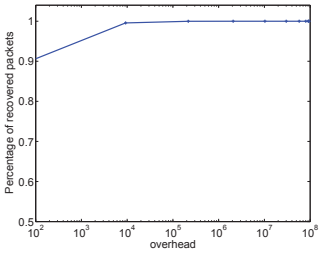

Fig. 3: Overhead with respect to recovered packets.

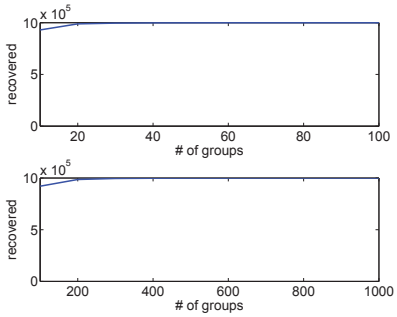

Fig. 4: Delays that can be calculated with respect to the number of groups for a total of 10^6 packets. Loss ratio $\rho = 10^{-5}$ for the upper figure and $\rho = 10^{-4}$ for the below figure.

information to the receiver would increase. For each group, there should be at least one augmented hash value. In the extreme case, a group contains only one packet and the number of augmented hash values is equal to the number of packets, leading to a high communication overhead. On the other hand, with more groups, the expected number of errors in each group will become smaller and thus the computation overhead for each group will be reduced. Therefore, we should carefully choose the number of groups considering the computation overhead and communication overhead.

Assume we divide the packets into g groups, we first calculate the probability for the number of errors in each group. Assume there are n packets and a total number of δ errors (including packet losses and reorderings) in a segment. It should be noted here that since packets are grouped based on a randomized hash function, thus here it does not matter for the distribution of the losses and reorderings. For example, losses and reorderings can be randomly distributed or be bursty. Denote $pr(err = \xi)$ as the probability that a particular group has ξ errors, we have

$$pr(err = \xi) = C(\delta, \xi) \cdot (\frac{1}{g})^{\xi} \cdot (1 - \frac{1}{g})^{\delta - \xi} \tag{8}$$

For ξ packet losses, the number of possible permutations we need to check in each group is $C(w, \xi)$. Further, for ξ packet reordering, the number of possible permutations is $d \cdot C(w, \xi)$. Intuitively, for a large shift in reordering, e.g., $d = 1000$, the computation overhead will be very large. However, by the grouping technique, the shift can be significantly reduced. For example, with 100 groups, the original shift of $d = 1000$ can be reduced to 1000/100 = 10 in average in each group, which significantly reduces the computation overhead. In the original data, the reordering shift of d is reduced to an average shift of d/g after dividing the data into g groups. We calculate the overhead as the number mappings in Algorithm 1 we need to test, which also refers to the number of loops (line 2 $\sim$ line 7) to be executed in Algorithm 1. We have

$$\begin{aligned} E(overhead) &\leq d/g \cdot \sum_{\xi=0}^{w} C(w, \xi) \cdot pr(err = \xi) \cdot g \\ &= d \cdot \sum_{\xi=0}^{w} C(w, \xi) \cdot pr(err = \xi) \end{aligned} \tag{9}$$

In practice, we select the number of groups proportional to the number of errors, e.g., in our implementation, we select the number of groups as $2\delta \sim 10\delta$. Usually, the number of errors is not high, thus the number of groups is also not high. Increasing the number of groups significantly reduces the computation overhead. To further reduce the number of errors, we can also leverage the sampling methods as in [5] [6] [7] [20] [21].

In our approach, considering that the number of errors should be very low in each group, we can ignore those groups with errors more than a threshold to reduce the overhead. We calculate the expected overhead $E(overhead')$ as

$$E(overhead') = d \cdot \sum_{\xi=0}^{k} C(w, \xi) \cdot pr(err = \xi) \tag{10}$$

where k is the maximum number of errors we consider in a group. With such a technique, there may exist groups that cannot be recovered due to errors more than k. For a larger k, there will be more recovered packets (packets that are not ignored). Meanwhile, the computation overhead will also accordingly increase. Thus we calculate the number of recovered packets with respect to computation overhead. Figure 3 shows the percentage of recovered packets with respect to computation overhead. The number of groups is set to 10δ. Compared with existing approaches based on a fixed sampling/probing overhead, the overhead of our proposed approach is related to the number of errors in each group. We can increase the number of groups when the errors increase. Since errors in most networks should be very low, the corresponding computation overhead should also be very low. As shown in Figure 3, the computation overhead is very low to recover most of the packets, e.g., the computation overhead is less than 10^5 to recover more than 99.99% of packets. It should also 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 would inevitably increase the communication overhead. Accordingly, the benefit of OPA becomes small.

It can also be seen that delays for most of the packets can be successfully calculated. For example, Figure 4 shows the number of packet delays that can be calculated when we only consider groups with less than two errors ($k = 2$). We can see that delay for almost all packets can be calculated.

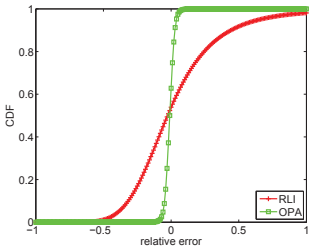

Fig. 5: comparison of relative per-packet delay error of RLI and OPA.

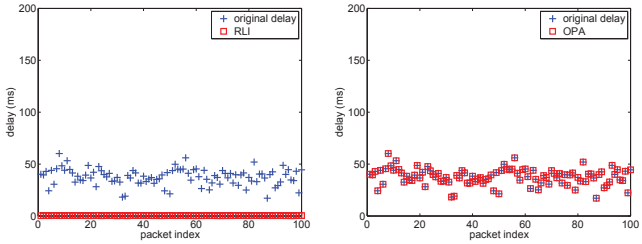

Fig. 6: left: true delay and the corresponding recovered large delays with RLI; right: true delay and corresponding recovered large delays with OPA.

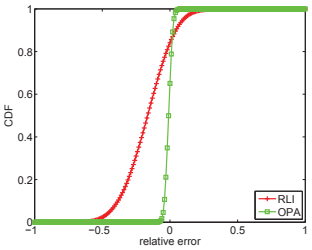

Fig. 7: error for estimation of large delays in RLI and OPA.

B. Communication overhead

We also consider the total communication overhead from the sender to the receiver. The communication overhead consists of two parts according to the two-layer design as shown in Figure 2. The first part is the overhead for sending OPAs. The second part is the overhead used for sending compressed time stamps. For the first part, according to our analysis in the above section, the overhead in each group is $(\xi + 1) \cdot L_1$ to calculate packet delay, where ξ is the number of errors in a group and L_1 is the size of OPA (12 bytes in our approach). More specifically, the total communication overhead for OPAs is $3L_1 = 3 \times 12 = 36$ bytes for groups 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 $3L_1 + \theta L_2$ for each group.

In our implementation, assume we have 100 groups for each segment, a compression ratio of $\theta = 1/10$ and a time stamp of 4 bytes, the total overhead is $100 \times 36 + 100 \times 10000 \times 4 \times 1/10 = 403600$ bytes for 10^6 packets, which can be sent in about 269 packets (with MTU = 1500 bytes). The amortized overhead is $269/10^6 \approx 3/10^4$. This also means for 10^4 packets, we only introduce no more than 3 packets to achieve fine-grained delay and loss measurement.

VII. Evaluation

A. Data set

As prior works [5] [6], we use the traffic data collected on real-world routers at San Jose (SANJ) and Chicago (CHIC) respectively. 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. We use the same model as in [7] [6] for delay distribution. Those two data sets and the settings are also used in other delay measurement approaches, e.g., [5] [7] [6]. As in those approaches, we add random losses and reorderings according to a specified loss and reordering rate that we introduce in each experiment.

B. Methodology

We implement the OPA data structure to calculate per-packet delay. For the time stamp compression part, there are many efficient wavelet compression methods [22]. We use the standard wavelet compression for all time stamps. Nevertheless, other wavelet compression methods can also be used. At the receiver side, we accordingly calculate per-packet delay by combining OPA and compressed time stamps. We evaluate the performance of the calculated per-packet delay in terms of per-packet relative delay error. We also demonstrate that our method can be used to detect abnormal delays that would otherwise unable to reveal.

To compare the performance of OPA with existing approaches, we implement the methods of LDA [5], FineComb [7] and RLI [6]. As for those methods only RLI can provide per-packet delay measurements, 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 that are ignored with RLI.

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

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

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 relative high rate 10^{-3}. As we have introduced, OPA is effective to detect abnormal delays (e.g., large delays). Thus we randomly add large delays to the delay distribution with a probability $1/10^4$. Since the data rate in our evaluation is relative 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 to that of $1/1000$. Thus we choose a rate of $1/1000$ in order to reduce the overhead of RLI.

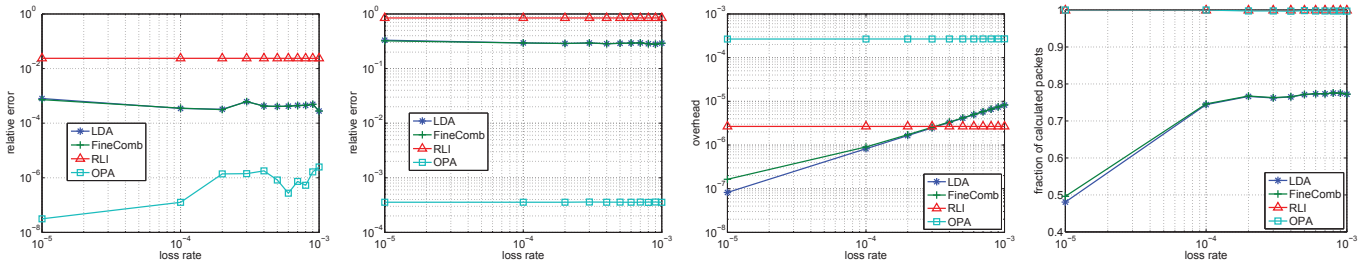

Fig. 8: Comparison of LDA, FineComb, RLI and OPA. (a) error of average delay, (b) error of standard deviation, (c) communication overhead and (d) fraction of delays that can be calculated.

For comparison, we define the per-packet relative delay error and average delay error. Assume there are totally n packets. Denote the true delay of packet i as $t_d(i)$ 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

$$\epsilon_i^F = (t_d^F(i) - t_d(i))/t_d(i) \tag{11}$$

Meanwhile, we define the average delay error as

$$\eta = (\bar{t}_d^F - \bar{t}_d)/\bar{t}_d \tag{12}$$

It should be noted that we can calculate per-packet relative delay error only for RLI and OPA. For other protocols, we cannot calculate the per-packet delay and thus the per-packet relative delay error. First, we compare the per-packet relative delay error provided by OPA and RLI. Figure 5 shows the CDF of per-packet relative delay error. There are two observations from this figure. First, the relative per-packet delay error for RLI is larger than OPA. Second, the relative per-packet delay error is very small for OPA. In our evaluation, most of the relative per-packet delay error is less than 5%. The averaged per-packet relative delay error is 2%.

To investigate how OPA can be used to effectively calculate per-packet delay, we compare the calculated delay by RLI and OPA with the original delay. In this experiment, we insert large delays to normal delays to see if RLI and OPA can correctly identify those large delays. As shown in Figure 6, the left figure illustrates the calculated delay with RLI and the original delay. We can see that most of the calculate delays with RLI are very small. This is because RLI uses interpolation based approach between reference packets. For such a method, since the large delays are infrequent, packets with large delays are not likely to be chosen as the reference packets. Therefore, the delay calculated from interpolation with reference packets of small delays will also be very small, resulting in a large estimation error for those large delays. This also explains Figure 5 that RLI has a high relative error. While in the right figure of Figure 6, we can see that OPA can effectively estimate each packet especially for those large delays with a small error. It should be noted in Figure 6, we only plot the large delays and their estimations for clarity.

To further investigate the performance, we calculate cumulative distribution of per-packet relative delay error for those large delays in Figure 7. In Figure 7, we show the cumulative distribution of delay error for RLI and OPA respectively. We have two findings. First, we can see that with OPA, per-packet delay can be efficiently recovered since most of the relative errors are distributed between -0.05 and 0.05. The relative per-packet delay error of RLI is larger than OPA. Second, with RLI those large delay are smoothed, leading to a negative relative error as shown in Figure 7.

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) in terms of average delay and standard deviation. Further, we also 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. In this experiment, we also randomly add large delays to the original delay.

Figure 8 (a) shows the comparison of the error for average delay with those four approaches. We can see that all those four approaches have a very small error for the average delay. For brevity, here we use the absolute value of relative error. Figure 8 (b) shows the comparison of error for standard deviation with those four approaches. We can see that all four approaches have acceptable relative error. RLI has the largest relative error. As we have explained, RLI cannot identify those large delays, leading to a larger error in average delay and standard deviation.

Figure 8 (c) shows the communication overhead from the sender to the receiver for delay calculation with those four approaches. It is worth noting that RLI and OPA calculate per-packet delay while LDA and FineComb calculate the aggregated delay statistics. The overhead of LDA is determined by the number of groups and banks. The overhead of FineComb is determined by the size of stash and the number of groups. The overhead of RLI depends on the number of reference packets. The overhead of OPA depends on the number of groups and the compression ratio. Figure 8 shows that the overhead of OPA and RLI is higher than that of the other two

approaches. This is reasonable since OPA and RLI provide per-packet measurements rather than aggregated delay statistics. Nevertheless, the overhead of OPA is still in the order of 10^{-4}, which indicates OPA only needs several packets to measure delay for 10000 packets. This is an acceptable overhead for practical use. The result also coincides with the analytical result in Section VI. It can also be seen that RLI has a relative stable overhead as long as the data rate and rate for reference packets are fixed.

Figure 8 (d) shows the fraction of recovered delays (i.e., delays that can be calculated for different approaches). In LDA and FineComb, a group of packets cannot be used if the group contains lost packets. The sampling technique can reduce the expected packet losses. However, it also reduces the total number of recovered delays. For 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. In practical implementation, 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. In practice, as shown in Figure 8 (d), by carefully selecting the group number, the fraction of recovered delays is very high for OPA and RLI, which is higher than that of the other two approaches.

VIII. Conclusion

Fine-grained delay measurement is critical to understand and improve system performance and diagnose network problems. We design a new data structure named order preserving aggregator (OPA) for fine-grained delay measurement. Based on OPA, ordering and loss information can be efficiently represented and recovered at the receiver with a small overhead by exploiting intrinsic data properties, i.e., most of received packets are order-preserved and correct. Leveraging OPA, we propose a two-layer design to efficiently calculate the per-packet delay. We implement OPA and evaluate its performance with real-world data. Results show that OPA achieves per-packet delay with 2% relative error while only incurring 0.04% overhead. We believe OPA can be widely used as an efficient per-packet delay and loss measurement approach in system management, performance monitoring and diagnosis.

Acknowledgement

This work is supported in part by NSFC under grant 61202359, 61373166, NSFC Major program under grant 61190110 and China Post doctoral Science Foundation under grant 2012M520013.

References

[1] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p-vod system," in *Proceedings of SIGCOMM*, 2008.

[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 *Proceedings of ACM SenSys*, 2012.

[3] R. MARTIN. [Online]. Available: Wall street's quest to process data at the speed of light. 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 Journal on Selected Areas in Communications (JSAC).*, vol. 21, no. 6, pp. 908–921, 2006.

[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 *Proceedings of ACM SIGCOMM*, 2009.

[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 *Proceedings of ACM SIGCOMM*, 2010.

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

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

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

[10] M. Lee, N. Duffield, and R. R. Kompella, "Maple: A scalable architecture for maintaining packet latency measurements," in *Proceedings of ACM SIGCOMM IMC*, 2012.

[11] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proceedings of the ACM SIGCOMM*, 2012.

[12] A. A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons, "Detecting the performance impact of upgrades in large operational networks," in *Proceedings of the ACM SIGCOMM*, 2010.

[13] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: Understanding the causes and impact of network failures," in *Proceedings of the ACM SIGCOMM*, 2010.

[14] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *Proceedings of the ACM SIGCOMM*, 2012.

[15] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proceedings of the ACM SIGCOMM*, 2011.

[16] "Ieee. standard for a precision clock synchronization protocol for networked measurement and control systems, 2002. ieee/ansi 1588 standard."

[17] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of USENIX OSDI*, 2002.

[18] A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss, "Near-optimal sparse fourier representations via sampling," in *Proceedings of STOC*, 2002.

[19] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "Fast, small-space algorithms for approximate histogram maintenance," in *Proceedings of STOC*, 2002.

[20] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," *IEEE/ACM Transaction on Networking*, vol. 9, no. 3, pp. 280–292, 2001.

[21] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," in *Proceedings of SIGCOMM*, 2004.

[22] O. Rioul and P. Duhamel, "Fast algorithms for discrete and continuous wavelet transforms," *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 569–586, 1992.