Probabilistic Network Telemetry

Benjamin Schaible, Kilian Holzinger*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: b.schaible@tum.de, holzinger@net.in.tum.de

***Abstract*—With ever increasing demand for large scale computing systems such as data centers, high-speed interconnecting networks too are becoming increasingly important. Measuring these networks is key to detecting and localizing performance issues. One important metric is packet latency. However, established network measurement systems are either not suitable for estimating packet latencies or introduce unwanted overhead [1].**

In this paper, we discuss three probabilistic approaches to estimating packet latencies, which were proposed in prior work: the Lossy Difference Aggregator (LDA), the Lossy Difference Sketch (LDS) and Reference Latency Interpolation (RLI). Following that, we compare these approaches against each other regarding their robustness to packet loss and reordering, their accuracy and the overhead they introduce. We come to the conclusion that, under low packet loss, latency estimates provided by the LDA and LDS are more accurate than ones provided by RLI.

***Index Terms*—probabilistic data structures, network measurement, high-speed networks**

1. Introduction

With ever increasing demand for large scale computing systems such as data centers, high-speed interconnecting networks too are becoming increasingly important. Such networks may consist of thousands of devices, switches and routers, making it nearly impossible to predict unwanted behavior. Network measurements have therefore become an essential tool to detect and localize network performance problems such as bottlenecks.

One of the key metrics measured is packet latency [1]–[3], which is the amount of time it takes for a packet to travel from one point to another in a network. Efficiently measuring packet latencies, however, especially in the order of microseconds, is a non-trivial matter. Established systems like NetFlow actively measure packet latencies by attaching timestamps to a sample of all observed packets [1]. High sampling rates are required in order for this to yield accurate average latency estimates, which in turn imposes a considerable network overhead [1].

In this paper, we discuss three probabilistic approaches to estimating average packet latencies, which were proposed in priror work. These approaches aim to provide accurate average latency estimates while remaning lightweight in terms of network utilization, memory usage and computational overhead. Namely, we discuss the Lossy Difference Aggregator [1] in Section 3.1, the Lossy Difference Sketch [2] in Section 3.2 and Reference Latency Interpolation [3] in Section 3.3. Following that, we compare their strengths and weaknesses in Section 4.

2. Probabilistic Data Structures

All of the probabilistic measurement approaches presented in this paper leverage their own application-specific data structures, though they share the same core concepts. We now discuss the most important ideas behind the probabilistic data structures used.

The data structures we are going to discuss in this paper are *sketches*. Sketches are used to *estimate* various metrics on streams of data. While they don't guarantee exact results, they usually come with a much smaller overhead when compared to perfectly precise measurements. Instead, they are *likely* to yield accurate results and therefore are being referred to as probabilistic.

A sketch usually aggregates some kind of value in a counter. In our case these values would be packet latencies. However, there usually is an unlikely but catastrophic occurrence which would *destroy* the counter, meaning that the value stored is skewed heavily and can no longer be used to obtain accurate estimates. In our case, this occurrence would be packet loss.

To deal with such occurrences, sketches use multiple counters and distribute samples evenly across them. This way, such a catastrophic occurrence would only *destroy* one of many counters, leaving the rest of them in a usable state. This even distribution is commonly achieved by calculating a hash of the sample and mapping that hash to an index within an array of counters. For example, assume that our sketch consists of n counters and we wish to insert the latency of a packet p. We may then calculate an index with $hash(p) \mod n$. Assuming the hash function used is *uniform*, meaning that each hash is produced with the same probability, this would result in an even distribution of samples across all n counters. In this paper, we refer to this technique as *hash partitioning*.

3. Measurement Approaches

Consider the following example: Two clients are exchanging a considerable amount of data via two switches (Figure 1). Now, let's say we want to measure the latency (or "delay") between these two switches, for example in order to gain insight into the state of the network. A simple way to achieve this is to have both switches attach a timestamp to each packet before dispatching it. The receiving switch may then calculate the latency by subtracting the packet's timestamp from its current time. This also allows one to calculate the average latency by summing up all measured delays and dividing by the number of delays measured.

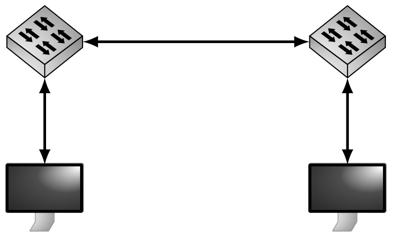
Figure 1: A simple measurement environment

The main problem with this approach is the overhead in bandwidth usage caused by the timestamps attached to each packet. Especially in data center networks, where hundreds of millions of packets are transmitted every second, even relatively small (32-bit) timestamps would quickly add up.

Additionally, any difference between clocks of the two switches will directly add to the error of the measured latency [1]. Therefore, both switches's clocks have to be synchronized tightly in order for the measurement to yield accurate results [1]. We are assuming microsecond-level clock synchronization, which may be achieved by making use of protocols such as IEEE-1588 [4].

We now discuss several probabilistic approaches to estimating the average latency between two network points.

3.1. Lossy Difference Aggregator

The Lossy Difference Aggregator (LDA), proposed by Kompella et al. in [1], is a probabilistic data structure, used to efficiently estimate the average and standard deviation of the latency between a sender A and a receiver B. The sender and receiver may be two network devices such as routers or switches, or two points within a single network device [1].

An LDA is used to conduct a *one-way* measurement, meaning that it estimates the average latency of packets being sent from A to B, ignoring packets sent the other way [1]. If a bi-directional measurement is desired, the measurement should be conducted twice, once in each direction.

3.1.1. Basic Idea. We now discuss the basic approach behind the LDA. Assume that A sends a total of N packets to B and we wish to measure the average latency of these packets. Let a_i be the time at which the i-th packet was dispatched from A, and b_i the time it was received at B, with $i \in \{1, \ldots, N\}$. For now, we assume there is no packet loss. Consequently, the latency of the i-th packet is $b_i - a_i$ and the average latency D corresponds to

$$D = \frac{1}{N} \sum_{i=1}^{N} (b_i - a_i) \tag{1}$$

$$= \frac{1}{N} \left(\sum_{i=1}^{N} b_i - \sum_{i=1}^{N} a_i \right) \tag{2}$$

as described in [1].

Recall the previously discussed approach of attaching a timestamp to each packet. In this context, A would attach to each packet its timestamp a_i, while B would count the number of packets it receives and add each packet's delay $b_i - a_i$ to an accumulator. The average delay could then be calculated using (1).

A Lossy Difference Aggregator allows one to conduct this type of measurement without the overhead of attaching timestamps to any packet. It operates on periodic time intervals of length T, with T usually being in the range from a few hundred milliseconds to a few seconds [1]. When using an LDA, instead of attaching timestamps to packets, we aggregate them both on the sender and receiver [1]. Then, at the end of each measurement interval, a control packet containing the sender's timestamp aggregate is sent to the receiver and used to calculate the average latency there [1]. The LDA is the data structure used to aggregate those timestamps, both at the sender and receiver [1].

In the context of the current example, consider a simplified LDA consisting of a single timestamp aggregator and a packet counter. We use one simplified LDA each for the sender and receiver. At the sender A, the simplified LDA then stores the sum $T_S = \sum_{i=1}^{N} a_i$ and the amount of packets sent N. At the receiver B, it stores the sum $T_R = \sum_{i=1}^{R} b_i$ and the amount of packets received R. Since we assume no packet loss, N and R are equal. As shown in [1], we can then transform (2) to use T_S and T_R:

$$D = \frac{1}{N} (T_R - T_S) \tag{3}$$

Using (3), we are able to calculate the average latency during a measurement cycle by building timestamp aggregates T_S and T_R at the sender and receiver respectively and by transmitting T_S to the receiver at the end of the cycle, assuming that $R = N$.

3.1.2. Dealing with Packet Loss. The simplified LDA however has a significant flaw: If even a single packet sent from A to B is lost, meaning that $N > R$, the receiver's timestamp accumulator becomes *unusable* for the remainder of the measurement interval and no meaningful average delay can be calculated. This is due to the fact that T_S would have aggregated more timestamps than T_R and that it is impossible to recover the original timestamps aggregated in T_S and T_R, eliminating the possibility of calculating the average of a smaller sample [1].

It is possible to mitigate the severity of this problem by making use of a simple measure proposed in [1]: Instead of a single accumulator-counter pair, use an array of m of these pairs, which we call a *bank* [1]. Then, for each packet, *hash partition* between the m accumulator-counter pairs in order to decide on which one to use [1]. Since this choice must be consistent across the sender and receiver, both must use the same hash function in order for this to work [1]. When using such a bank of m accumulator-counter pairs, a single packet loss would only corrupt a single one of these pairs while the others remain usable [1]. The average delay of a subset of all sampled packets may then be determined by combining all usable accumulator-counter pairs [1].

However, using a bank only allows the LDA to handle low packet loss rates [1]. A high loss rate, combined with a high amount of throughput, would still be likely to quickly corrupt the entire bank [1]. To cope with this, instead of sampling every packet received, a fixed sampling probability p can be imposed on the bank with the goal of reducing the number of potentially *unusable* accumulator-counter pairs [1]. Since, in order to maintain consistency, a packet should be sampled at the receiver if and only if it was sampled at the sender, it is logical to use the same sampling rate at both ends [1]. A common hash function may then be used on each packet in order to apply this sampling rate [1]. Based on the rate of packet loss, a suitable sampling rate p, which maximizes the expected sample size, may then be determined [1].

In a realistic scenario however, the packet loss rate is usually unknown and prone to change over time [1]. This can be dealt with by using an array of n banks, each one tuned to a different packet loss rate through its sampling rate p_i [1]. When using multiple banks, it is advantageous for those banks to have *disjoint* sampling sets, meaning that no packet is ever sampled by more than one bank, since this opens up the possibility of combining all usable accumulator-counter pairs across all banks in order to calculate the average latency from the greatest possible sample, without the possibility of some packets being counted multiple times [1]. If the sampling probabilities $p_1, \ldots, p_n$ were to be powers of $\frac{1}{2}$, disjoint sampling sets could easily be achieved by hashing each packet to a bitstring, where each bit has an equal probability of being 0 or 1 [1]. Then, use the number of leading zeroes of the bitstring to determine which bank, if any, will sample the packet [1].

3.1.3. The complete Data Structure.

The full LDA consists of an $m \times n$ matrix of accumulator-counter pairs. Each of the n colums represents a separate *bank*, holding its own disjoint set of samples [1].

The update procedure when sampling a packet, as proposed in [1], is:

1) Calculate a uniform hash of the packet.
2) Use the hash and the sampling probabilities $p_1, \ldots, p_n$ to decide whether the packet should be sampled and which bank to use in that case.
3) Use the hash to select an accumulator-counter pair of the chosen bank.
4) Add the timestamp at which the packet was received to the timestamp accumulator and increment the packet counter.

At the end of each measurement interval, the sender's LDA is transmitted to the receiver (or vice versa) and the average latency is estimated [1]. Let T_A and T_B be the sum of all usable timestamp accumulators of the sender's and receiver's LDAs respectively [1]. A timestamp accumulator-counter pair is considered usable if its packet counter at the receiver matches the corresponding one at the sender [1]. Let S be the effective sample size, which is the sum of all usable packet counters at the receiver (or the sender) [1]. Following from this, the average delay estimate D is

$$D = \frac{1}{S}\left(T_B - T_A\right) \tag{4}$$

as described in [1].

3.2. Lossy Difference Sketch

The Lossy Difference Sketch (LDS) is a probabilistic data structure proposed by Sanjuas et al. in [2]. It builds on the basic ideas behind the LDA (Section 3.1) and is meant to be used to estimate the average packet latency between a sender and a receiver, while being lightweight in terms of memory usage and computational overhead. Just as the LDA, it is used to conduct *one-way* measurements [2]. However, unlike the LDA, which estimates the average latency of *all* packets sent from a sender to a receiver, it produces *per-flow* estimates, meaning that a separate latency estimate can be obtained for each flow observed [2]. In this context, we consider a *flow* a tuple of the source and destination address, source and destination port and the protocol [2].

The motivation behind the LDS is based on the observation that packet latencies might differ considerably between flows, meaning that average latencies over all packets may not prove sufficient in order to detect and analyze application-specific network performance issues [3].

3.2.1. Basic Idea.

Much like an LDA, the LDS estimates average latencies by aggregating timestamps and comparing them in periodic time intervals [2]. However, in order to conduct *per-flow* measurements, it needs to distinguish between flows. It does so in a probabilistic manner by *hash-partitioning* between timestamp accumulator-counter pairs based on the flow f [2]. Additionally, each timestamp accumulator-counter pair also stores a *flow digest* [2] in order to detect potential inaccuracies caused by packet reordering between the sender and receiver [5]. We refer to a tuple consisting of a timestamp aggregator, packet counter and flow digest as a *bucket* [2].

3.2.2. Functionality.

The LDS consists of a $R \times C$ matrix of *buckets* [2]. Each packet is sampled once per row [2].

Sampling Procedure. When sampling a packet, for each row, an index within that row is determined based on a hash of the flow f and row index i [2]. Then, the index is offset by a number up to k, determined by hash of the full packet, resulting in an even distribution of samples across k adjacent buckets, k being an adjustable parameter of the LDS [2]. The bucket at the resulting index is then updated, meaning that the packet's timestamp is added to the accumulator, the packet count is incremented and the flow digest is XOR-ed with a hash of the full packet [2]. Inspired by the LDA, this index shift is done to reduce the impact of packet loss and, in this case, packet reordering by distributing samples of a flow across k different buckets [2]. We note that this allows for potential collisions between samples of flows [2] and will discuss how the LDS attempts to find the buckets with the least interference in order to accurately estimate an average delay. Further, it is evident that all hash functions used must be equivalent on the sender and receiver as to maintain consistency [2].

Estimating Average Delays. In order to estimate the average latency of a flow f, first select all usable buckets of the ones samples of f were collected in [2]. A bucket is considered usable if both its packet count and its flow digest at the sender and receiver match [2]. Then, find the bucket with the lowest packet count n [2]. We are assuming this is the one with the least interference from other flows [2]. Then, from the previously determined set of usable and related buckets, select those that have sampled at most $n(1 + \alpha)$ packets, where α is a configurable parameter, and calculate the average latency by combining them [2]. Reference [2] suggests using a value of 0.1 for α.

3.3. Reference Latency Interpolation

Reference Latency Interpolation (RLI), proposed by Lee et al. in [3], is an alternative technique to estimating *per-flow* packet latencies between a sender and a receiver. It leverages the observation that packets belonging to different flows tend to experience similar latencies when sent with little delay in between each other [3]. Just like the previously discussed approaches, RLI is used to conduct *one-way* latency measurements [3].

When using RLI, the sender periodically sends a reference packet holding a timestamp to the receiver and the receiver then uses the timestamp to calculate the reference packet's delay [3]. For every non-reference packet received, its delay is then estimated using linear interpolation between the previous and the next reference packet's delays as well as factoring in the individual packet's size and the link capacity [3]. Since a packet's delay can only be estimated after another reference packet was received, all information needed to estimate its delay, namely its flow identifier and timestamp of its arrival, are stored in a buffer until the delay can be estimated [3].

It should be mentioned that the decision of when to inject a reference packet is not straightforward. For example, injecting a reference packet every n packets sent could work well under high network load but might result in too few packets being injected under low load, lowering the accuracy of the estimate [3]. Injecting reference packets in fixed time intervals instead would solve the accuracy issues for low network load but in turn might result in too few packets being injected under high load [3]. Reference [3] therefore suggests a combination of both rules, effectively adapting the injection rate dynamically based on the network load.

4. Comparison

We now compare the discussed approaches in Table 1. The properties we are comparing them against are the type of measurement (overall or per-flow estimate), robustness against packet loss and reordering, network overhead and accuracy.

4.1. Robustness

While the LDA and LDS make use of *hash partitioning* in order to deal with packet loss, potentially losing a portion of their samples, RLI samples every packet that was not lost [1]–[3]. The LDA assumes packets arrive in the same order they were sent and has no protection against packet reordering [1], [5], while the LDS detects and avoids affected samples [2] and RLI is not affected by it since it does not aggregate timestamps [3].

TABLE 1: Comparison between LDA, LDS and RLI

Type	**Measurement**	**Loss**	**Reord.**	**Overh.**	**Acc.**
LDA	overall	+	–	++	++
LDS	per-flow	+	+	++	++
RLI	per-flow	++	++	+	+

Measurement: Overall or per-flow measurement
Loss: Robustness against packet loss
Reord.: Robustness against packet reordering
Overh.: Network overhead
Acc.: Accuracy of measurement/estimate

4.2. Network Overhead

Both the LDA and LDS do not incur any network overhead during their measurement intervals and transmit the data structure once at the end of each measurement interval [1], [2]. Since RLI regularly injects reference packets, which also affect the forwarding behavior of routers, we consider its overhead slightly larger [3].

4.3. Accuracy

Evaluations of the LDA and LDS in [1] and [2] respectively have shown that both suffer a very low mean relative error under low ($< 1\%$) packet loss. However, since different flows may interfere with each other within an LDS, it tends to be more accurate for larger flows than smaller ones because large flows carry more weight within their latency estimate [2]. When facing packet loss rates below 5%, the LDA suffers a reasonably small mean relative error of 3% to 9% [1]. RLI, on the other hand, experiences a mean relative error of 10 to 12% for moderate to high link utilization and around 30% for low link utilization [3]. A direct comparison between the LDS and RLI in [2] showed that the LDS outperforms RLI in terms of accuracy under low ($< 1\%$) packet loss.

5. Conclusion

In this paper, we have discussed three different probabilistic approaches to estimating packet latencies between two network points. First, we have discussed the basic idea behind aggregating timestamps and how the Lossy Difference Aggregator (Section 3.1) leverages it. We then proceeded with the Lossy Difference Sketch (Section 3.2), which makes use of the basic ideas behind the LDA, but estimates *per-flow* latencies [2]. Finally, we have looked at Reference Latency Interpolation (Section 3.3), an alternative approach to estimating *per-flow* latencies by interpolating between latencies of reference packets [3]. Following that, we have compared the three approaches in Section 4, where we have seen that both the LDA and LDS usually provide more accurate latency estimates than RLI, though they are less robust against packet loss and reordering.

References

[1] R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese, "Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 255–266, 2009.

[2] J. Sanjuàs-Cuxart, P. Barlet-Ros, N. Duffield, and R. R. Kompella, "Sketching the delay: tracking temporally uncorrelated flow-level latencies," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 483–498.

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

[4] J. C. Eidson, M. Fischer, and J. White, "Ieee-1588™ standard for a precision clock synchronization protocol for networked measurement and control systems," in *Proceedings of the 34th Annual Precise Time and Time Interval Systems and Applications Meeting*, 2002, pp. 243–254.

[5] M. Lee, S. Goldberg, R. R. Kompella, and G. Varghese, "Fine-grained latency and loss measurements in the presence of reordering," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 2011, pp. 329–340.