An Online Framework for Flow Round Trip Time Measurement

Xinjie Guan[1]*, Xili Wan[1]*, Ryoichi Kawahara[2], Hiroshi Saito[2]
[1]University of Missouri - Kansas City, Kansas City, MO 64110, USA
[2]NTT Network Technology Laboratories, NTT Corporation, Tokyo 180-8585, Japan

***Abstract*— With the advent of high speed links, online flow measurement for, e.g., flow round trip time (RTT), becomes difficult due to the enormous amount of requirements on computational resources. We address this problem by proposing the double-deletion bloom filter (DDBF) scheme, which alleviates potential hash collisions of a standard bloom filter by explicitly deleting used records and implicitly deleting out-of-date records. As a result, it accurately estimates the current TCP flow RTT distribution with limited memory space, even with the appearance of multipath routing and Syn flooding attacks. Theoretical analysis indicates that the DDBF scheme exhibits desirable accuracy with a constant and small amount of memory. We further validate our scheme using real traces and demonstrate significant memory-saving without degrading accuracy.**

I. Introduction

With the rapid increase in traffic and the change in traffic patterns due to the widespread use of various types of applications [1], understanding traffic characteristics/quality-of-service (QoS) through network measurement is crucial for network management. Round trip time (RTT) is an important QoS metric for network management [2], [3]. From the viewpoint of network operations, it is essential to monitor the current QoS status, such as the x percentile of RTTs in a network, and to detect changes in QoS to promptly identify the cause of QoS degradation. A ping command is generally used to measure the RTT between a specific pair of hosts for trouble shooting, rather than for continuously monitoring the QoS status of flows in a network. On the other hand, the RTTs of flows can be estimated offline from packet trace data measured in a network by using tcptrace [4]. However, with the advent of multi-gigabit links, e.g., OC-192 or OC-768, computational resources will soon be exhausted if packet processing and recording are performed for individual TCP flows passing through the network. Thus, per-flow RTT measurement becomes challenging, especially for online use.

We therefore propose a scheme called double-deletion bloom filter (DDBF) for on-line flow RTT measurement. The DDBF scheme can accurately estimate the flow RTT distribution with constant and small computational space through explicitly and implicitly removing useless records and releasing occupied bits in the bloom filter (BF). Compared to existing solutions, our scheme significantly reduces the computational cost while maintaining the same accuracy. Thus it is particularly appropriate for online measurement on a high-speed link with high packet rate and numerous concurrent TCP flows.

Various sampling methods have been proposed for scalable flow measurement. Among them, a hash-based sampling method samples and tracks flows whose identifier hashes to a values less than a predefined threshold. This enables us to reduce the number of flows to be observed and avoids the bias on large flows caused by random packet sampling [5]. To reduce the inaccuracy caused by hash collisions, the application of a BF, which consists of multiple hash functions, has also been studied [6]–[8].

However, most papers focus on measuring individual flow size or the total number of flows [5]–[10], but not on flow RTT estimation. These methods cannot be directly applied or are not appropriate for flow RTT measurement. For example, for flow size measurement, accumulated counters are used [9], while for flow RTT measurement, we have to calculate the difference in arrival times between associated packets. Another difference is that the size of a flow can be estimated only after observing the entire flow, while the flow RTT can be quickly obtained by using the Syn-Ack method [2]. In other words, the measuring time for each flow in RTT measurement is much shorter than that in measuring each flow size, which makes it possible to save memory space for flow RTT measurement by sharing memory space used for maintaining intermediate per-flow records among multiple flows during different times.

This paper is organized as follows. We first describe the problem of applying a BF to RTT measurement in Section II, before proposing our DDBF scheme and the intuition behind the design in Section III. We also examine its efficiency in this section through theoretical analysis. We compare the accuracy and efficiency of the DDBF scheme with two other schemes by using real traces in Section IV. Finally, we give our conclusions in Section V.

II. Bloom Filter for RTT Measurement

A. Problem Statement

We consider a measurement point such as a router between source and destination nodes, where packets typically arrive and leave at a very high rate. The flow RTT is indicated by the time difference between two matched packets in the TCP 3-way handshake phase [2]. Our aim is to measure the RTTs of passing TCP flows by tracking the first Syn-SynAck packet-pair. A flow is associated with a 5-tuple **flow ID**:

*This work was done while the authors were at NTT Service Integration Laboratories.

source IP, source port, destination IP, destination port, and direction (upstream or downstream). Every packet in this flow is associated with the Flow ID.

For on-line RTT measurement, we need to record the arrival time for each arriving Syn packet and match its corresponding SynAck packet. This requires a large amount of memory space and computational cost due to the large number of flows. For online RTT measurement, a scheme that can answer an RTT query at any time with limited time and space is necessary.

B. Flow RTT Measurement with Bloom Filter

The RTT of a TCP flow can be estimated as the time difference between the first pair of Syn and SynAck packets from this flow. Therefore, to estimate the RTT distribution for all flows passing by a certain measurement point, we need to record the arrival time of Syn packets and map SynAck packets to their corresponding Syn packets. The most straightforward (**SF**) scheme is to maintain a record for each arriving Syn packet in an array, in which each entry contains a key (a 5-tuple flow ID) and a value indicating the Syn packet arrival time. When a SynAck packet arrives at t_{sak}, we retrieve the corresponding Syn arrival time t_{Syn} using the Flow ID of that SynAck packet and calculate the RTT of this flow as $t_{sak} - t_{Syn}$. However, the SF scheme is not scalable with a significant amount of flows.

Another natural scheme is to apply a BF to organize Syn arriving records and assist query operations. A BF consists of a bitmap of b one-bit cells, all of which are initially set to 0, and a group of k independent hash functions with range from $\{0, 1, \cdots, b-1\}$. To use a BF to compute flow RTTs, we extend each cell in the bitmap from one bit to s_t bits to record the Syn arrival time, where s_t can be adjusted to trade off accuracy and space efficiency, as further discussed in Section IV.

When a Syn packet arrives at t_{Syn}, we hash the associated flow ID with k hash functions and obtain k hash results, each of which indicates a cell index in the BF. We then update the timestamps in the corresponding cells with t_{Syn}. The timestamps of this Syn packet may be overwritten by later arriving Syn packets due to hash result collision. Nevertheless, as long as one of the k corresponding cells is not overwritten by later arriving Syn packets, we can consider the arrival time of this Syn packet as the oldest timestamp in the k corresponding cells. Therefore, on the arrival of a SynAck packet, if all associated cells are not empty, the RTT of this flow is the difference between the oldest timestamp in the k corresponding cells and this SynAck packet's arrival time. In contrast, if any of the k corresponding cells is empty, this SynAck packet definitely comes from a new flow, and we simply drop it.

In Fig. 1, we demonstrate the above scheme of applying the straightforward BF (**SBF**) to RTT measurement. In this figure, we label each packet with its arrival time. A Syn packet arrives and its arrival time t_1 is recorded in cells 1, 4, and 14 according to the hash results of three independent hash functions. When the first SynAck packet arrives at time t_2, we check its corresponding cells, and find that the value of cell 6 is 0. We conclude that this SynAck packet does not belong to any pre-existing flow and drop the packet. Later, another Syn packet arrives at t_3 and overwrites the timestamp of the first Syn packet at cell 4. At time t_4, a SynAck packet arrives whose corresponding cells are all occupied by timestamps. Therefore, the RTT of the flow to which this SynAck packet belongs can be estimated as $t_4 - t_1$, the arrival time of this SynAck packet minus the oldest timestamp among the three corresponding cells.

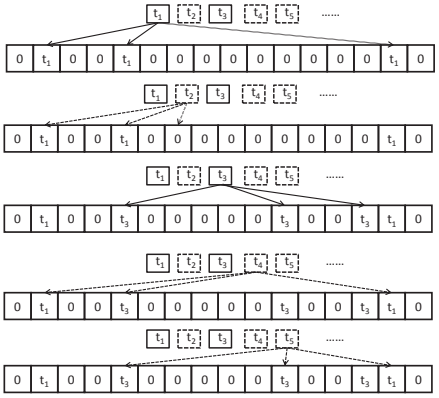

Fig. 1. Bloom filter for RTT Measurement: Solid-lined rectangle represents Syn packet while dashed-lined rectangle represents SynAck packet, and label t in each rectangle indicates arrival time of each packet. These packets are arranged in chronological order.

C. Discussion

It is well known that a BF may induce a false positive by falsely considering un-inserted elements as inserted because of hash collisions. When applying a BF to RTT measurement, the SBF scheme may underestimate RTT if all the timestamps in a Syn packet's BF record are overwritten by later arriving Syn records before its corresponding SynAck packet arrives. The BF may also misestimate RTT by mismatching *orphan SynAck* packets resulting from multi-path routing with irrelevant Syn records. For example, the packet at t_5 in Fig. 1 is an orphan SynAck, unrelated to the Syn packet at t_1, meaning that $t_5 - t_1$ is not a valid RTT estimate[1]. Because of the existence of multi-path routing, some SynAck packets do not pass by the same measurement point with their corresponding Syn packets.

Table 1 lists the number of arrived Syn packets, SynAck packets, and matched Syn and SynAck pairs in some traces. Data Sets A and B are two passive, anonymized packet traces collected from a router providing Internet access to residential users. The details of these data sets are discussed in Section IV. As observed in Table 1, more than $2/3$ of Syn packets in Data Set A and more than $1/2$ of Syn packets in Data Set B never match with any SynAck packet, and more than $1/2$

[1]We found that the probability of the underestimation is relatively small while the probability of the misestimated RTT being larger than τ is calculated as $1 - (1 - e^{-\frac{1}{b}\lambda\tau})^k$ where λ is the flow arrival rate, and such probability can be large. Detailed analysis is omitted due to space limitation.

Table 1

	No. of Arrived Syn Packets	No. of Arrived SynAck Packets	No. of Matched Syn and SynAck Pairs
Data Set A	1,450,649	767,895	383,050
Data Set B	80,208,552	40,421,824	39,877,651

of SynAck packets in Data Set A are orphan SynAck packets whose corresponding Syn packets have not passed by the same measurement point. This may be due to the existence of Syn-flooding attacks or to multi-path routing. These unmatched Syn records gradually accumulate in SBF, resulting in potential errors when estimating flow RTTs. Because of false positives in the BF, the orphan SynAck packets may be falsely matched with unrelated Syn records leading to misestimation of flow RTTs. It may be even worse in large-scale networks, where more routers are deployed between an access network and the Internet.

III. Proposed method

A. Algorithm

One intuitive approach for reducing the effect of un-matched Syn packets is to gradually fade out the records that have stayed in the BF for a long time. However, it burdens computational resources to periodically scans the BF cells for out-of-date records. Thus, we introduce the double delete bloom filter (**DDBF**) which *explicitly* deletes records that have been matched with arrived SynAck packets and *implicitly* removes out-of-date records that have not been matched with any SynAck packets within a certain time. In contrast with a periodical scan, both deletions in our DDBF scheme can promptly and efficiently remove useless and expired records. In DDBF, each cell is composed of a s_t-bit timestamp and a s_c-bit counter that store the arrival time and number of Syn packets, respectively.

On the arrival of a Syn packet, we hash its flow ID with k hash functions and obtain k hash results, each of which corresponds to a cell in the DDBF. In each of the k corresponding cells, the timestamp is overwritten with this Syn packet's arrival time, and the counter is increased by 1.

On the arrival of a SynAck packet, we first hash its flow ID to obtain the corresponding cells in the DDBF. We then calculate the difference between this SynAck packet arrival time t_{sak} and the oldest timestamp t_{ts} in the corresponding cells. If the time difference is smaller than a given time span vt, we consider this time difference as the RTT of the flow and decrease the counters in all corresponding cells by 1 (*explicit deletion*). Otherwise, we simply discard the received SynAck packet and decrease the counters in the corresponding cells with timestamps older than $t_{sak} - vt$ by 1 (*implicit deletion*).

B. Analysis

We theoretically discuss the memory efficiency of the D-DBF scheme by comparing it with the space consumption of the SBF scheme when they are both required to achieve the same accuracy. Since, according to observation of real traffic trace data, the inaccuracy is mainly from falsely matching orphan SynAck packets, we only consider false positives from orphan SynAck packets. We denote the number of inserted elements in a BF by N and the bitmap length by b.

We assume both SBF and DDBF schemes share the same parameter settings in terms of the number of hash functions k and arriving Syn packets n. We also assume a fraction $\omega(\omega < 1)$ of passing flows can be deleted explicitly or implicitly. Therefore, the number of Syn records in the DDBF scheme is $N_v = (1-\omega)n$ while that in the SBF scheme is $N_s = n$. The false positive rate of a BF is approximately

$$fpr(N,b) = (1-(1-\frac{1}{b})^{kN})^k \approx (1-e^{-kN/b})^k. \quad (1)$$

To achieve the same false positive rate in the DDBF and SBF schemes, $fpr(N_v, b_v)$ should be equal to $fpr(N_s, b_s)$ where b_v and b_s are bitmap lengths in the DDBF and SBF schemes, respectively. Thus, we obtain

$$(1-\omega)b_s = b_v. \quad (2)$$

Therefore, to obtain the same accuracy, the DDBF scheme removes ωb_s cells. Considering that each cell contains a one-bit label and a s_t-bit timestamp, the total space consumed by the SBF scheme is $b_s(1+s_t)$. The DDBF scheme extends the one-bit label in the SBF scheme to a s_c-bit counter in each cell. A $s_c = 4$-bit counter is used similarly in the counting BF (**CBF**), an alternative scheme to diminish false positives by deletion operations [11]. Therefore, the total space consumed by the DDBF scheme is $b_v(4+s_t)$. By substituting b_v with $(1-\omega)b_s$ according to Eq. (2), the total space consumed by the DDBF scheme can be represented as $(1-\omega)b_s \times (4+s_t)$. Thus, the ratio of memory consumed by the SBF scheme to that consumed by the DDBF scheme is

$$\frac{b_s(1+s_t)}{(1-\omega)b_s \times (4+s_t)} = \frac{1+s_t}{(1-\omega)\times(4+s_t)} \quad (3)$$

In practice, we use the $s_t = 20$-bit timestamp, which is large enough to distinguish different arrival times in $2^{10} \times 2^{10} \approx 15$ min at millisecond granularity. From Eq. (3), we conclude that, as long as $\omega > 12.5\%$, the DDBF scheme exhibits the same accuracy as that of the SBF scheme with a smaller amount of memory space. Moreover, by calculating the ratio of matched Syn and SynAck pairs to arrived Syn packets in Table 1, we find that around 26% (resp. 50%) of records in Data Set A (resp. B) can be explicitly removed.

Note that the records can be further implicitly removed using the DDBF scheme if it is not updated in a time span vt, bring further potential memory saving. Section IV shows that the DDBF scheme is more accurate than the SBF and CBF schemes even with much smaller memory space[2].

IV. Evaluation

We based our study on passive, anonymized packet trace data sets measured at two different points, Data Sets A and

[2]As for computational cost, both of the DDBF and SBF schemes can prevent cost increase when N increases because they only access k cells based on the hashed results regardless of the size of N.

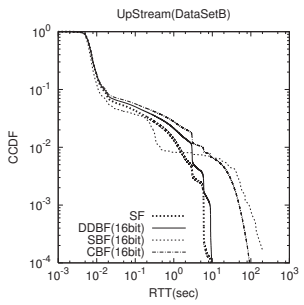

Fig. 2. RTT distribution estimated using four schemes

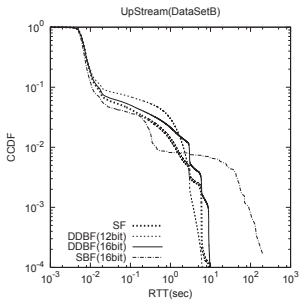

Fig. 3. RTT distribution when we change memory size

B. Each measurement point is the aggregation point from where residential users access the Internet through Fiber-to-the-Home (FTTH). Each data set was measured in April 2009. The average traffic volume in Mbps for Data Set A, measured for 1-hour, was 213.8 Mbps in the upstream direction and 208.5 Mbps in the downstream direction. The total number of TCP flows was 915,081 in the upstream direction and 991,877 in the downstream direction, representing 93.08% and 94.85% of total traffic in the upstream and downstream directions, respectively. As for Data Set B, a 24-hour trace, the average traffic volume in Mbps was 273.3 Mbps in the upstream direction and 470.2 Mbps in the downstream direction, while the number of TCP flows was 73,605,701 in the upstream direction and 54,972,746 in the downstream direction, representing 77.86% and 92.16% of total traffic volume, respectively.

We assessed the accuracy of the DDBF scheme using the trace data and compared it with the SBF and CBF schemes, where the CBF scheme corresponds to the DDBF scheme without implicit deletion. Fig. 2 shows the complementary cumulative distribution function (CCDF) for flow RTTs of Data Set B. The accuracies of the SBF, CBF, and DDBF schemes are assessed by comparing them with the results measured using the SF scheme. That is, we used the results of the SF scheme as ground truth. We chose 16 bits as the size of the hash result (i.e., $b = 2^{16}$ cells), and used $k = 3$ hash functions and a $s_t = 20$-bit timestamp. We set vt to 10 s according to the observation that almost all flow RTTs are shorter than 10 s [12].

As shown in Fig. 2, with the same number of hash functions and the same size of hash results, the SBF scheme performed worst, due to serious hash collisions caused by the large number of inserted elements. Even though the CBF scheme performed better thanks to the removal of records that will not be used later, the DDBF scheme outperformed the CBF scheme as it implicitly deletes out-of-date records so that we can maintain the minimal number of real effective records. Due to space limitation, we only present the results of Data Set B in the upstream direction. However, we also tested our scheme on the other traces, and the same trends hold.

By comparing the accuracies of the SBF and DDBF schemes under different space constraints, we found that the DDBF scheme saves memory space for the same accuracy, as shown in Fig. 3. The cell numbers of the SBF and DDBF schemes in this figure are $b = 2^{16}$ and $b = 2^{12}$, respectively, and each cell in the SBF scheme contains a 20-bit timestamp while each cell in the DDBF scheme contains a 20-bit timestamp and a 4-bit counter. Therefore, the SBF scheme requires $2^{16} \times 20$ bit ≈ 1.3 Mb, while the DDBF scheme only consumes $2^{12} \times (20 + 4)$ bit ≈ 98 Kb. (Note that, as shown in Fig. 2, the CBF scheme is inferior to the DDBF scheme even when the CBF scheme uses $b = 2^{16}$, which corresponds to $2^{16} \times (20+4)$ bit space. This means that the CBF scheme requires much more memory than the DDBF scheme with $b = 2^{12}$.) Furthermore, Fig. 3 demonstrates that the DDBF scheme is more accurate even with much smaller memory space as we proved in Section III-B.

V. Conclusion

For accurate and efficient flow RTT measurement on high-speed links, we proposed DDBF scheme. This scheme explicitly deletes used records and implicitly deletes out-of-date records, so that the number of inserted elements in the DDBF is reduced, resulting in less hash collisions and potential errors. Theoretical analysis and real trace experiments show that our DDBF scheme exhibits high accuracy with a relatively small and constant memory space.

References

[1] C. Labovitz, S. I.-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. *ACM SIGCOMM 2010*, pages 75–86, 2010.

[2] H. Jiang and C. Dovrolis. Passive estimation of tcp round-trip times. *ACM SIGCOMM Computer Communication Review*, 32(3):75–88, 2002.

[3] B. Veal, K. Li, and D. Lowenthal. New methods for passive estimation of tcp round-trip times. *Passive and Active Network Measurement*, pages 121–134, 2005.

[4] http://www.tcptrace.org/.

[5] N Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking (TON)*, 9(3):280–292, 2001.

[6] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter braids: a novel counter architecture for per-flow measurement. In *Proc. ACM SIGMETRICS international conf. Measurement and modeling of computer systems*, pages 121–132, 2008.

[7] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani. Fast monitoring of traffic subpopulations. In *Proc. 8th ACM SIGCOMM conf. Internet measurement*, pages 257–270. ACM, 2008.

[8] J. Sanjuas-Cuxart, P. Barlet-Ros, and J. Solé-Pareta. Counting flows over sliding windows in high speed networks. *NETWORKING 2009*, pages 79–91, 2009.

[9] P. Tune and D. Veitch. Towards optimal sampling for flow size estimation. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 243–256. ACM, 2008.

[10] Giuseppe Bianchi, Nico Nico d'Heureuse, and Saverio Niccolini. On-demand time-decaying bloom filters for telemarketer detection. *ACM SIGCOMM Computer Communication Review*, 41(5):5–12, 2011.

[11] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8(3):281–293, 2000.

[12] Srinivas Shakkottai, R. Srikant, Nevil Brownlee, Andre Broido, and kc claffy. The rtt distribution of tcp flows in the internet and its impact on tcp-based flow control, 2004.