An Efficient Approach to Per-Flow State Tracking for High-Speed Networks

Brad Whitehead and Chung-Horng Lung
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
{bwhitehe, ehlung}@sce.carleton.ca

Peter Rabinovitch
Alcatel-Lucent
Ottawa, Canada
peter.rabinovitch@gmail.com

Abstract — Maintaining per-flow information and state is a crucial topic in network monitoring. Tracking per-flow state is a relatively new area. Two main approaches have been proposed for tracking state: Binned Duration Flow Tracking (BDFT) and Fingerprint-Compressed Filter Approximate Concurrent State Machine (FCF ACSM). BDFT which uses Bloom filters is time efficient, whereas FCF ACSM using d-left hash tables has near-perfect memory efficiency but has higher computational cost. This paper presents a hybrid method (BDFT-H) by employing the best features of BDFT and FCF ACSM to achieve both time and space efficiency. Performance analysis and comparisons are conducted for BDFT, FCF ACSM, and BDFT-H. These methods are all intended for implementation on high-speed routers where resources such as memory and CPU time are limited. For the computational performance of the three schemes, we find that based on analysis, d-left hashing may require substantially more computational resources than Bloom filters. We also conduct simulations to compare the accuracy of these three schemes and the results show that all three methods can achieve over 99% accuracy on traces of real traffic. The proposed approach provides the best overall tradeoff between time and space efficiency.

Keywords — network monitoring; flow tracking, Bloom filter, high-speed networks

I. INTRODUCTION

Monitoring the transmission of various network protocols is crucial to ISPs, as better network control leads to better utilization which leads to lower costs. A common task to network measurement applications is to store some amount of state about individual flows. Monitoring per-flow state on high-speed routers requires an approach that is highly efficient in terms of both memory and processing capabilities. Packet sampling (e.g., 1 in 20 sampling) normally returns low accuracy. For long duration and high bandwidth flows, a naive implementation can manage to track some long duration flows. However, the memory usage could be very high [13].

This paper proposes a method of tracking per-flow state that is both time and space efficient. The main idea is to integrate two efficient approaches; one is efficient in time, the other one is highly effective in memory usage but has higher computational cost.

The first method, Binned Duration Flow Tracking (BDFT), was proposed [13,15] as an efficient method for flow tracking. In brief, BDFT operates by grouping individual flows into “bins” which represent the current state of the flow. BDFT assigns time ranges to each state (bin) (e.g., 0-15sec, 15-45sec, 45-75sec, 75-105sec), and moves flows to the next time range (state) on a periodic basis. BDFT inherits much of its time and space efficiency from the use of counting Bloom filters [3] as the data structure which represents the bins.

Fingerprint-Compressed Filter Approximate Concurrent State Machine (FCF ACSM, [3]) is another approach that can effectively track per-flow duration. FCF ACSM is memory-efficient but has higher computational cost than BDFT.

This paper presents a hybrid of BDFT and FCF ACSM, which takes advantage of the best characteristics of each called BDFT-H. Computational performance is a metric that is rarely analyzed in determining a method's performance in this area. Computational analysis is an important part of the overall performance picture of each method, because CPU time is critical for high-speed routers. Computational comparison of the three methods is central to this paper. In addition, the accuracy and memory usage of each method is determined by running simulations with real-world traffic traces.

The rest of the paper is organized as follows: Section II describes the background. Section III highlights BDFT and FCF ACSM. Section IV presents BDFT-H. Section V discusses computational analysis of these three methods. Section VI presents the experiments and results. Finally, Section VII concludes our study.

II. BACKGROUND

Tracking per-flow state is a relatively new area with little work. Bloom filters and its variants such as counting Bloom filters [2,4] have become wide-spread in network monitoring. The main reason is their ability to provide a time and space efficient data structure to represent a set of items when some errors are acceptable [2]. Some other Bloom filter variants are Space-Code Bloom filters [13] and Time-Decaying Bloom filters [7] which can track flow duration with medium accuracy. It is shown that a Bloom filter can be implemented in hardware and can scale to OC-192 (10Gbps) speeds [1].

Bonomi, et al. [3] presented two variations of a state tracking system using Bloom filters. Their first method uses a single counting Bloom filter to store a set of <flow, state> pairs. Their second approach uses counting Bloom filters to store both a count and a state in each cell corresponding to a flow's hashes. The main difference between their methods and ours is that this paper introduces the concept of using multiple bins (and therefore Bloom filters) to achieve high accuracy with low computational requirements.

Cuckoo hashing requires only a constant number of items to be moved for each insertion, depending on the load of the hash table. However, standard cuckoo hashing suits software applications, not high-speed routers [9]. In [10], the authors designed a scheme that allows at most one item to move during...
insertion, which results in higher space utilization. Both [9] and [10] consider the availability of content addressable memory that allows parallel lookups.

SCD [14] is method of filtering network traffic such that only fully established TCP flows will pass through the filter. Once a TCP SYN has been detected from both sides of a connection, SCD will report that the connection was successfully established. In [14], we have shown that filtering unsuccessful flows can reduce the processing requirements by 95%. In this paper, SCD is adopted to pre-filter incomplete TCP flows for all three methods that we compare.

III. BDFT AND FCF ACSM

As stated in Section I, this paper is primarily based on two approaches: BDFT and FCF ACSM. Therefore, the following describes both methods in detail.

A. BDFT

Network operators desire to track the state of network flows to extract additional information about the traffic on their network. This information can be varied, and the required state transitions for a flow can be equally varied. Hence, flexibility is important, but flexibility comes at a price as arbitrary state transitions may not be practical on today’s router.

It is observed that many flows share a common state and state transitions happen for many flows at the same time. The flows therefore can be intuitively thought of as grouping flows into “bins”. A bin represents a group of flows that are all in the same state. And since all state transitions are unitary and happen at the same time for all flows in a bin, state updates can be performed by simply moving all of the flows in one bin to another. Such a simple model of state transitions creates some advantages for a method which tracks flow state.

BDFT [13,15] is a data structure and algorithm designed to track the approximate duration of all TCP flows seen on a high-speed router. Using BDFT as a classier allows the long duration flows to be aged for further processing by a DPI device to determine if they are P2P traffic or other traffic types.

In BDFT, counting Bloom filters were selected as the default bin data structure. This selection allows the intrinsic operation of Bloom filters to be used to our advantage, by replacing the flow identification information with hashes. Figure 1 shows the basic operations that define how BDFT maintains duration information for all flows.

1) Add a Flow

Flows are added to Bin #1 when they enter a “partially established” state, which we define as receipt of a SYN packet from either side of a connection (1st or 2nd step of the TCP 3-way handshake). Flows are added by creating k hashes from the flow identification information, searching all bins to see if the flow already exists; and if not, incrementing the counters in Bin #1 corresponding to those hashes. Searching all bins can be avoided by using SCD pre-filtering as described in Section 2, so flows are only added on the 2nd step of the handshake.

2) Remove a Flow

TCP packets containing a FIN or RST flag signals the end of a flow, at which point the flow is removed from its bin. Flows are removed by searching from the shortest-duration bin to the longest. When the flow is found, the counters corresponding to the flow’s hashes are decremented. The counters corresponding to the flow must be decremented every time a FIN or RST is received, until one of the counters reaches zero. Bins are searched starting with the youngest based on the observation that 50% of flows last less than 2 seconds, and 90% last less than 45 seconds [6], so the flow will most likely be found in the youngest bin.

3) Aging

As time advances during the operation of BDFT, the flows in a bin become older, until the oldest flow in the bin is older than the time range (duration) of the bin, at which point the bin must be aged. The aging process is key to BDFT, it allows the maintenance of the state for all flows. By keeping flows in counting Bloom filters, and aging the filters in time, no flow-specific information such as flow start time needs to be kept. When a bin is expired, the flows that are currently in the bin are moved to the next longer duration bin.

4) Search for a Flow

Searches are performed starting with the oldest bin first using k hashes, and moving to sequentially younger bins, until a bin is found (or a false positive is found). The reason is based on the assumption that the longer duration bins are the least likely to generate a false positive. The duration for the flow is an estimate calculated by determining the midpoint time for the bin the flow was found in. For example, if the flow is in the 45-75sec bin, a duration of 60sec would be returned.

An example of BDFT operations is presented here. BDFT has bins with time ranges of 0-15sec, 15-45sec, 45-75sec, and 75-135sec. For instance, if a flow lasts for 55 seconds and the flow arrives just after Bin 1 was aged, so the flow will be in Bin 1 for its full 15 second duration. The following steps illustrate the BDFT operations taking place:

- The new flow arrives; its hashes are calculated based on IP Src/Dst, Port Src/Dst, and protocol type
- The flow is added to Bin 1 by incrementing the counters corresponding to the hashes
- After 15 seconds Bin 1 expires and its flows are moved to Bin 2
- After an additional 30 seconds Bin 2 expires and its flows are moved to Bin 3
- After 55 seconds from the flow start, a TCP FIN is received for the flow, and the removal process begins
- The flow's hashes are calculated as above
- The Bins are searched from the flow’s hashes starting with Bin 1
- The flow is found in Bin 3, so the counters corresponding to the hashes are decremented in Bin 3
B. FCF ACSM

Bonomi, et al. [3] present three methods of tracking per-flow state, of which the most promising is FCF ACSM. The FCF ACSM employs a d-left hash table [5] as the main data structure, which is augmented with additional fields to store state and fingerprint conflict information. FCF ACSM was found to be very accurate and to have good memory efficiency when compared to the Bloom filter based approaches. Its accuracy remains good up to ~80% memory efficiency (depending on table/bucket/cell configuration), after which bucket overloading can become a problem. The FCF ACSM can be easily adapted to track the duration of network flows, and therefore provides a good measure to compare the computational and accuracy performance of BDFT against.

To track flow duration with FCF ACSM, the bins of BDFT are assigned numbers and those numbers are stored in the state field of FCF. For example, if there are 14 bins in BDFT then a 4 bit state field would be created in FCF, and one state assigned to the “don’t know” state. New flows are inserted with state zero, and then in each aging cycle all of the flows in the FCF are checked to see if their current state needs to be updated. Removal of flows occurs with the standard d-left method of checking each cell in the selected buckets for the fingerprint.

D-left hash tables have a number of advantages and disadvantages when compared to counting Bloom filters, these are summarized here and explored on later in this paper. The primary advantage of d-left hashing is that it comes close to the performance of a perfect hash. This leads to an even distribution of items in the buckets of the tables, which means very efficient memory usage when the size of the set of items to be stored is known in advance. Since the size of the set is rarely known, the size of the hash should be overestimated; d-left hashing shares this minor disadvantage with BDFT. The near-perfect hashing of d-left comes at the usual price: reduced memory usage is traded off for higher computational requirements. Section V shows that d-left hashing requires several times more computational power than Bloom filters.

IV. BDFT HYBRID–BLOOM AND D-LEFT

Observing the computational performance of BDFT, and the near-perfect hash and memory efficiency of FCF ACSM, leads us to explore a combination of the two schemes which takes advantage of the best characteristics of each. In the hybrid scheme (BDFT-H), a number of the older bins in BDFT are replaced with a single FCF ACSM. This arrangement takes advantage of the fast Bloom filters for the very frequent flow state, of which the most promising is FCF ACSM. The memory efficiency and accuracy of FCF ACSM for the long-lived but seldom changing flows.

The number of initial short-duration bins that are left as counting Bloom filters should be determined by analysis of flow duration distribution and the available memory resources. Once a flow reaches the end of the chain of counting Bloom filters, it is moved into a FCF ACSM and set to state zero. Removal of flows occurs as normal in BDFT, with the younger filters being checked for the flow first, and then the FCF.

One twist in BDFT-H occurs when moving the flow into the FCF, as this process requires more hash bits than are available. Aging the flows into the FCF is also somewhat complicated by the fact that the Bloom filter does not contain any specific information on each flow or allow a hash for each flow to be easily determined. To get around these problems each of the hashes for the counting Bloom filter is associated with a separate table (instead of having all hashes modify counters in the same table), so when aging to the FCF takes place any counters that are greater than zero in table zero indicate active flows that should be moved to the FCF. The position of the counter in the table can give some indication of the hash of the associated flow. For example, if the table is 65536 elements then sixteen bits of hash information will be available from the table. If this is not a sufficient number of bits then this one table can be expanded to include extra bits of the hash along with the counter. When a flow is added to the first Bloom filter, the table zero extra hash bits are set to the upper bits of the flow’s hash. If a hash collision results (the counter in table zero is already at 1 or greater) then the extra hash bits can be stored in a small backup table.

Figure 2 depicts the life of a flow as it passes through a BDFT-H flow tracker. Here, each Bloom filter table is 16 bits in size, and the extra bits table holds 8 bits. There are two Bloom filters before the FCF, and the FCF consists of 3 tables, each with 256 buckets (8 bits) and a 16-bit fingerprint. The flow is initially added to Bin 0 with counter addresses {9876, 8765, 4321} and 8 extra hash bits from hash zero which are 123. The flow is then moved to bin 1 during the first aging process. The next aging iteration requires that the flow is moved from a Bloom filter to the FCF. In this example, 8 bits are required for a bucket address (123) and 16 bits are required for the fingerprint (4321). Note that only one bucket (least loaded and the smallest is the tie breaker) is needed for each flow in FCF. These requirements correspond with the number of counters (16 bits) and the extra bits (8 bits), although in other setups there is no need to have the number of bits correspond, so long as the total number of bits available from the Bloom size plus the extra bits equals the number required for the bucket address plus the fingerprint size.

Selection of the parameters for the Bloom filters and the d-left hash can proceed as for BDFT and FCF ACSM. The size of the d-left hash can be much smaller as it only needs to store a relatively small number of long duration flows.

Figure 2. Example of a flow in BDFT-H
comparison, several assumptions about the environment and implementation can be made. First, we assume that they are implemented in software. Next, assume that they are executed in a single thread on a processor that performs sequential memory access only. Third, ignore the effects of caching. In this case a fairly accurate performance measure can be obtained by counting the average number of memory access required and the number of branch instructions required.

To provide a fair comparison, a few more points must be accounted for. Note that for both methods the first step in every operation is to calculate the hashes for the item, since this step is common to all operations it will not be included in this analysis. For d-left hashing, it is assumed that one hash function is used, and is permuted to obtain the hashes for the other tables, and there are four tables with six cells per bucket. As well, for d-left hashing there is no suggested way to store the cells in the buckets.

For example, an array with an active/inactive flag for each cell, or a doubly-linked list would work, but both carry additional computational requirements. D-left hashing implemented with the array-based scheme requires checking every cell on every operation, so the worst case performance discussed below becomes the average performance, and maintaining a doubly-linked list requires extra pointer operations. For BDFT, it is assumed that three hash functions are used and counters are sized large enough that they do not overflow. Note that we deliberately choose a number of hash functions less than optimal to save on the number of memory accesses (the lower number of hash functions also reduces the probability of a counter overflow, so smaller counters can be used). The performance of BDFT-H is assumed to be similar to BDFT except where noted otherwise, since the majority of the operations involve insertion and removal of short-lived flows.

We can now explore the operations of insert, remove, search, and age. Results of the following discussion are summarized in Table I.

**Insertion** of elements into BDFT requires modification of the first bin only. The current counters corresponding to the hashes must be read, incremented, and then updated with the new values, and a check performed for overflow. This process requires 3 memory reads and 3 writes, and 3 comparisons (for 3 hash functions). D-left hashing requires that the buckets corresponding to the hashes be searched for the incoming flow, this requires 6×4 = 24 reads and compares (in our example with 6 cells/bucket). Once the flow is confirmed not to be present, the flow must be added to the least-filled bucket with tie-breaks to the left, requiring another 4 comparisons and a memory write. D-left also requires a check for overflow and false positive, and if either is true requires further logic; however, these are not counted due to being rare occurrences.

**Removal** of items from BDFT requires searching the bins starting with the shortest duration bin first. Given a nominal BDFT bin setup for fine-grained monitoring of nominal Internet traffic, about 75% of searches will end with the first bin, about 15% will end in the second, and some flows will require searching all bins [6,13]. We take the expected case to be two bins. Searching one bin requires 3 memory reads and 3 comparisons. D-left hashing must search 6×4 = 24 buckets in the worst case to find the flow, i.e., 24 reads and comparisons, with an average of 12. Once the flow is found, d-left must update the doubly-linked list or the flag the cell inactive.

**Searching** for an item in BDFT starts with the longest duration bin. Searching is the most expensive operation in BDFT, with the worst-case requiring reading the counters corresponding to the hashes in every bin except the shortest. This would require 3×(number of bins - 1) memory reads and the same number of comparisons, with about half the bins being the average case.

The requirements for d-left hashing are the same as insertion and removal, 24 reads and comparisons, with 12 being the expected case. Note that searches on active flows are relatively rare occurrences that are initiated by a human operator or software agent on a specific flow.

**Aging** of flows must be performed during BDFT or FCF ACSM operation. Depending on the implementation, aging bins may require merging counting Bloom filters of different sizes, without overwriting the data in the recipient filter. In this case, the merging process requires reading all of the counters in both filters, adding them together, and writing the results to the longer-duration filter. In special cases the aging process can be optimized. For instance, if both filters are the same size, and the recipient filter can be overwritten (because its contents were just aged to a longer duration bin), then aging the filter is a simple pointer update. For d-left hashing with state maintenance, every cell in the buckets must be read, a decision or comparison made, and its state updated. BDFT-H can be designed such that all aging between Bloom filters requires only a pointer update and a clear of one of the filters. Aging flows from the Bloom filters to the d-left hash for BDFT-H is a more complex operation, requiring reading entries from the Bloom filter and multiple d-left insert operations.

Aging occurs every 15 seconds, or as configured. In a nominal BDFT configuration, only the shortest duration filter needs to be processed every 15 sec, and the second shortest duration filter every 30 sec, and so on for all filters. The longer duration filters need to be processed only every 5-10 min.

To calculate relative performance it is assumed that the first Bloom filter in BDFT is of size 1000 and so is the second filter, and a merge operation must take place. The d-left filter is assumed to be of size two times the first Bloom in BDFT (2000 entries), and has 500 active flows. For the Bloom to d-left aging for BDFT-H, the Bloom filter size is 1000 entries with 150 active flows, requiring 1000 reads for the filter and 150 inserts into a d-left with 3 tables and 5 cells/bucket.

In summary, as shown in Table I, BDFT provides very good computational performance for the most frequent operations. D-left requires about 3.3 times more operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Operation</th>
<th>Mem. Reads</th>
<th>Mem. Writes</th>
<th>Branches</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDFT</td>
<td>Insert</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>FCF ACSM</td>
<td>Insert</td>
<td>24</td>
<td>1</td>
<td>29</td>
<td>54</td>
</tr>
<tr>
<td>BDFT</td>
<td>Removal</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>FCF ACSM</td>
<td>Removal</td>
<td>12</td>
<td>1</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>BDFT</td>
<td>Search (rare)</td>
<td>21</td>
<td>6</td>
<td>21</td>
<td>42</td>
</tr>
<tr>
<td>FCF ACSM</td>
<td>Search (rare)</td>
<td>12</td>
<td>0</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>BDFT</td>
<td>Aging (periodic)</td>
<td>2000</td>
<td>1000</td>
<td>1000</td>
<td>4000</td>
</tr>
<tr>
<td>FCF ACSM</td>
<td>Aging (periodic)</td>
<td>2000</td>
<td>500</td>
<td>2000</td>
<td>4500</td>
</tr>
<tr>
<td>BDFT-H</td>
<td>Aging (periodic)</td>
<td>1</td>
<td>1 + memset</td>
<td>0</td>
<td>2 + memset</td>
</tr>
<tr>
<td>BDFT-H</td>
<td>Aging (to d-left)</td>
<td>3250</td>
<td>150</td>
<td>3000</td>
<td>6400</td>
</tr>
</tbody>
</table>
(54+25) than BDFT (9+15) for an insert-remove pair (one flow), but offers greater search performance (24 vs. 42 operations for BDFT). BDFT-H combines the best attributes from Bloom filters and d-left hashing, offering fast insert-remove of short lived flows, and quick search times for long-duration flows.

VI. EXPERIMENTAL ANALYSIS

A. Trace Characteristics

Two traces of Internet traffic were used, with the two traces hereafter referred to as C_04 (CAIDA [12]) and N_12 (NLANR [8]). Both traces represent one hour of Internet traffic. They were selected to demonstrate performance over long periods of time. The average bandwidth in both traces is roughly similar at 100Mbps and 200Mbps, sufficient to demonstrate performance on high-speed links. TABLE II summaries the characteristics of these two traces.

<table>
<thead>
<tr>
<th>Trace Characteristics for TCP 5-tuple Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Flows</td>
</tr>
<tr>
<td>Total Established Flows</td>
</tr>
<tr>
<td>Average Active Flows</td>
</tr>
<tr>
<td>Timed Out Flows</td>
</tr>
<tr>
<td>Unique IPs</td>
</tr>
</tbody>
</table>

The distribution of flow durations is an important factor in the design of a BDFT filter array, as described in Section III. The duration distribution allows a rough estimation of the required size of the bins in BDFT, and therefore an estimation of the total memory usage of the BDFT array. One of the few papers to discuss the number flows that last specific lengths of time is “Dragonflies and Tortoises” [6], where they find that 75% of flows last less than two seconds.

B. Experimental Setup

To determine the accuracy of the three methods, we implemented an experimental framework that compares the duration recorded by a perfect flow tracker with the estimated flow duration reported by the each of the methods. The perfect flow tracker was implemented using standard per-flow tracking and measurement techniques. We defined a flow to be the standard 5-tuple of IP source and destination address, TCP source and destination port, and protocol type (TCP only).

As mentioned earlier, packets from the trace are also passed to a SCD filter [14] to all three methods as well as a simple FIN/RST-checking Bloom filter. Mitigating timeout effects is still an open area of research, so for our implementation, we automatically remove flows after two minutes with no activity, as adopted in TCP standard.

To present the accuracy of tracking duration with bins, we adopt the following definition of a success. A tracking success is defined to be an estimated flow duration result that is within 50% of the actual flow duration, for flows that are greater than 30 seconds in duration. This definition is based on the design goals of BDFT; it is intended to track long-duration flows to an approximate duration. Tracking failures occur if a false negative or don’t know is returned, or the duration is outside the acceptable range (mostly caused by a false positive).

For BDFT, the basic bin sizing allocates 128k entries for the first bin, which we refer to as the base filter size. The base size uses 180224 bytes of memory. We tested BDFT at multiples of the base filter size, and the same multiples of all the other filter sizes to analyze the performance of BDFT at various memory usage levels.

The FCF ACSM was implemented using standard arrays to implement the tables, buckets and cells [3]. An empty cell was indicated with a fingerprint of 0, and therefore any item that had a natural fingerprint of 0 was set to have a fingerprint of 1. To match the number of states tracked by the bins in BDFT, 4 bits were allocated for state (for 16 possible states, 15 of which are used). A 2-bit counter is maintained for each cell to detect false positives on insertion and maintain the count of actual items stored (a count of one indicates two total items for the fingerprint). When a false positive occurs on inserting, the state for that cell is set to “don’t know” only if the state of the inserted item is different from the state of the cell.

Hybrid BDFT uses bloom filters based on the timings for the first four filters of normal BDFT. All four Bloom filters are the same size to increase the computational efficiency of BDFT-H. An extra hash table is used with sufficient bits stored to meet the requirements of the FCF ACSM. The extra hash table is the size of base Bloom filter, and has a backup table 5376 bytes in size to resolve hash conflicts.

C. Experimental Results

TABLES III-V show the performance of BDFT, FCF ACSM, and BDFT-H, respectively, with various memory configurations. D-left parameters shown in the tables represent d/b/h/f used in [3], where d is the number of hash functions, b is the number of buckets, h is the height of each bucket (cells), and f is the size of fingerprint in bits. Performance for all three of the methods was close to theoretical expectations. All three methods were able to achieve practical accuracy performance at memory usage levels low enough to be practical for implementation on high-speed routers. The distinguishing factors between the three methods are the memory used to achieve a desired performance level and the computational requirements of the method. The BDFT-H method is seen to provide the best accuracy/space/time tradeoff for most applications. The remainder of this section discusses modifications to the basic simulation to obtain our results and specific insights into the performance of each of the methods.

<table>
<thead>
<tr>
<th>Trace Characteristics for TCP 5-tuple Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Flows</td>
</tr>
<tr>
<td>Total Established Flows</td>
</tr>
<tr>
<td>Average Active Flows</td>
</tr>
<tr>
<td>Timed Out Flows</td>
</tr>
<tr>
<td>Unique IPs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trace</th>
<th>Memory Usage</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_04</td>
<td>90112</td>
<td>95.46%</td>
</tr>
<tr>
<td>C_04</td>
<td>180224</td>
<td>99.19%</td>
</tr>
<tr>
<td>C_04</td>
<td>360448</td>
<td>99.87%</td>
</tr>
<tr>
<td>C_04</td>
<td>720896</td>
<td>99.97%</td>
</tr>
<tr>
<td>N_12</td>
<td>2816</td>
<td>96.85%</td>
</tr>
<tr>
<td>N_12</td>
<td>5632</td>
<td>99.79%</td>
</tr>
<tr>
<td>N_12</td>
<td>11264</td>
<td>99.98%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trace</th>
<th>Memory Usage</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_04</td>
<td>4/1024/6/16</td>
<td>67584</td>
</tr>
<tr>
<td>C_04</td>
<td>4/1024/9/16</td>
<td>101376</td>
</tr>
<tr>
<td>C_04</td>
<td>4/2048/6/16</td>
<td>135168</td>
</tr>
<tr>
<td>C_04</td>
<td>4/4096/6/18</td>
<td>294912</td>
</tr>
<tr>
<td>N_12</td>
<td>4/64/4/12</td>
<td>2304</td>
</tr>
</tbody>
</table>

978-1-4244-5637-6/10/$26.00 ©2010 IEEE
Both BDFT and FCF ACSM have two issues that can lead to buildup of orphan and stale flows in the tables. These issues are false positive removal (FPR) and flow timeouts. For our simulations, the effect of these issues is removed by accurately correcting for FPR and removing flows from the filters that have not seen activity for two minutes. These corrections provide increased insight into the mechanisms that affect performance for the three methods. Without the correction, the accuracy reduces significantly for smaller memory usage [15]. Also, note that a simple way to avoid FPR errors is to oversize the tables such that the chance of false positives is small. The use of SCD as a pre-filter reduces the number of flows that are timeouts by 95% on average [14].

BDFT accuracy results are in-line with our expectations. With only 0.257 bits of storage required per flow, BDFT achieves 99.87% accuracy on the C_04 trace, and with only 0.128 bits of storage required per flow BDFT achieves 99.79% accuracy on the N_12 trace. Bits per flow is crucial to high-speed routers and is calculated by dividing the memory usage (e.g., 360,448 bytes × 8 for C_04 shown in TABLE III for 99.87% accuracy) by the number of established flows (11,215,873 for C_04 shown in TABLE II). These results also show an important point; in practice it is not necessary to use 4 bits/counter in a counting filter, which has been the standard number; in these simulations 2 bits/counter was sufficient. Note that the primary failure mechanism in overload conditions (e.g., 90112 bytes for C_04) is counter overflow. These results confirm that BDFT is a good choice to track flow state when CPU time is more expensive than memory.

FCF ACSM accuracy results are in-line with our expectations. With only 0.096 bits of storage required per flow FCF ACSM achieves 99.95% accuracy on the C_04 trace, and only 0.064 bits of storage required per flow for 99.90% accuracy on the N_12 trace. As memory decreases, the primary failure mode for FCF ACSM was bucket overflow. These results confirm that FCF ACSM is a good choice to track flow state when memory is more expensive than CPU time.

The accuracy/space/time tradeoff for BDFT-H is such that it provides the best overall performance of the three methods. With only 0.214 bits of storage required per flow, BDFT-H achieves 99.94% accuracy on the C_04 trace, and with only 0.286 bits of storage required per flow BDFT-H achieves 99.86% accuracy on the N_12 trace. An interesting trend to note with BDFT-H is that the per-flow memory requirements for N_12 are higher than for C_04, this is the reverse of BDFT and FCF ACSM. This difference is caused by the overhead required to maintain the extra hash bits for the Bloom to d-left translation. This overhead becomes relatively smaller as the number of bits that can be obtained from the BDFT section increases (with increasing Bloom filter size). The results confirm that BDFT-H is the best choice for almost all applications, except if the memory costs are very high.

VII. CONCLUSION AND FUTURE RESEARCH

This paper presented BDFT-H for per-flow state tracking. Three methods were thoroughly analyzed to determine computational performance, and simulations were run with two real-world packet traces to determine memory usage and accuracy. The best performing method was BDFT-H based on the “binning” concept of tracking flow state in combination with a FCF ACSM. The other two methods met performance expectations, with BDFT displaying good computational efficiency, and FCF ACSM having good memory efficiency. The “binning” concept appears to offer good performance for tracking per-flow state for a specific class of state machines.

The problems of FPR and timeouts when tracking flow state are open problems which require further research.

REFERENCES


