A Resilient and Collaborative Replacement Framework for Per-Flow Monitoring

Gunwoo Nam, Pushkar Patankar, Seung-Hwan Lim, Bikash Sharma, George Kesidis, and Chita R. Das
Department of Computer Science and Engineering
The Pennsylvania State University
Email: {gnam, patankar, seulim, bus145, kesidis, das}@cse.psu.edu

***Abstract*—In the context of a collaborating surveillance system for active TCP sessions handled by an networking device, we consider two problems. The first is the problem of protecting the flow table of per-flow monitoring from overflow and the second is developing an efficient framework estimating the number of active flows and identifying ”heavy hitter” TCP sessions, to reduce the overhead of data exchanged. Our approaches are sensitive to the limited hardware and software resources allocated for this purpose in the linecards in addition to the very high data rates that modern line cards handle; specifically we are interested in cooperatively maintaining a per-flow state with a low cost which has resiliency on dynamic traffic mix. We investigate a traditional timeout processing mechanism to manage the flow table for per-flow monitoring efficiently, called Timeout-Based Purging (TBP), our “clock”-like flow replacement algorithms using a straightforward replacement policy, called “clock”, and finally hybrid approaches combining these two. The effectiveness of our approaches on Internet traces is demonstrated. Our experiments show that our schemes can significantly reduce both false positives and false negatives regardless of both the conditions of flow table occupancy, even under SYN flooding. Our hybrid schemes estimate the number of active flows accurately, and confine the heavy hitters without additional state.**

I. Introduction

Modern networking devices, *e.g.,* routers, firewalls and Network Intrusion Detection Systems (NIDS), have the capability of monitoring tens of thousands of TCP sessions handled by them [8], [26]. The goal of such monitoring, which is called “stateful inspection” in the context of the firewall system [7], could be to detect attacks or intrusions or to deter end-systems that opt-out of communal (and voluntary) TCP congestion control mechanisms. In lieu of an active response, detected TCP session anomalies can be forwarded to corresponding NIDS if there is a collaborating structure within an enterprise network under the same authority. For the NIDS, tracking more complex flows (e.g., spanning multiple correlated TCP sessions) may be desirable. In the latter case, there is a reasonably clear response option even for routers deployed in the core: TCP sessions that are deemed unresponsive to congestion can be quarantined and rate-limited to their “fair” share of available bandwidth resources. Traffic monitoring, QoS assurance and usage-based billing [4] also motivate per-flow monitoring of TCP flows. Response in a QoS context could be to somehow protect or prioritize service for persistently congestion-responsive flows in packet memory, and quarantine or throttle-back persistently unresponsive flows [32].

On the other hand, the overhead of maintaining the per-flow monitoring in network devices, *i.e.,* wire-speed packet processing and corresponding updates [18], is high. A large number of short-lived flows or abnormal flows results in significant consumption of the flow memory as well as an increase in the volume of data exchanged for collaboration. Even though a SRAM implementation of the flow table has became typical for fast operations despite its limited capacity [6], [33], this has worsen the scaling problem. If the flow table is dynamically sized using shared system memory, this resource can be exhausted in an extreme situation. If the flow table size is fixed, the per-flow monitoring system cannot perform correctly when a flash crowd of short-lived flows or Denial-of-Service (DoS) attacks occur. Prediction of the “proper” size of flow table is not trivial because the per-flow monitoring should adapt to the dynamic traffic mix without reference to whether the table utilizaion is low or high. For this reason, the per-flow monitoring itself could be targeted by mallicious attacks, *e.g.,* SYN flooding or port scanning, which could result in performance degradation of the networking device and the network connected to it [15], [23].

To remove or disable entire set of stale (not recently touched) flows from memory, time-outs can generally be managed for flow purging using a “touch bit” at the flow level [6], [31]. However, setting up a time-out threshold [19], [15] trades-off protecting flow memory and prematurely purging flows. If time-out is too short, it is less vulnerable but active flows could be falsely removed. Conversely, If time-out is long, it increases the number of flow entries. The use of a wide range of timeout values, *e.g.,* 3600 seconds in an IDS [27] to 15 seconds for a flow measurement tool [1], indicates the difficulty in choosing timeout values.

A flow is bidirectional traffic, which may take different paths in two ways: reverse path may be different from forward path and OSPF/iBGP reroute is needed in the context of resiliency to failure. To monitor these flows properly, a distributed IDS system, such as Symantec ManHunt [30] or ISS SiteProtector [14], might be preferred to cooperate within an enterprise level. To make a per-flow monitoring for the distributed IDS, information sharing of high volume of raw data become a concern. If you have a framework to approximately cluster the flows somehow, we can report these specific groups of flows or a summary data for each cluster to avoid heavy exchange of data. We will not explicitly focus on IDS issues herein except to evaluate performance in the presence of SYN floods. Note that an attack response strategy to SYN floods may be more effectively and feasibly enacted by intermediate routers instead of those very close to the victim or attacking end-systems. Our culminating empirical results in this paper compare the performance of different approaches under SYN flooding.

This complex performance and scalability issues motivate us to devise a method to fully utilize the flow table regardless of traffic mix, *i.e.*, protect routers maintaining per-flow state from flow table overflows in the presence of SYN floods (or also flash crowds of flows) and preserve as many idle flows as the maximum capacity of the flow table permits. To achieve this, we suggest a replacement framework with low cost for maintaining per-flow state. First, we investigate a "clock"-like Flow Replacement (CFR) employing a simple and fast replacement policy, called "clock" [5]. In this CFR, touch bit is set based on "clock", but based on purging. It performs similar to Least Recently Used (LRU) algorithm without the overhead of maintaining the sorted list of flows. By extending the touch bit to two bits and giving incremental state for flows, we can discriminate several groups of flows based on the number of incoming packets, *e.g.*, SYN flooding packets, short-lived flows and long-lived flows. This multiple bits allows multi-level replacement based on clustering, which means scan-resistant to port scanning or SYN flooding []Bansal04. Also we can benefit from the flow groups by generating the summary of clusters or exchange a particular one only. We then investigate the hybrid approaches combining CFR and a traditional purging scheme, which reduce the volume of the flow table by removing the outdated flows periodically. This hybrid approaches can improve to estimate the number of active flows and confine heavy hitters into the highest state. Compared with tiditional purging schemes, our replacement framework can give much resiliency regardless of traffic mix. Our preliminary analysis shows that most cases a flow lookup can be performed within a memoery access to find an unused entry, although our CFR suffers from a exhaused search in the worse case, *i.e.*, when the flow table is full.

We evaluate the performance of our algorithms using three sets of Internet traces colleted from different networks. We compare their performance with two state-of-the-art algorithms, namely Finger-Compressed Filter (FCF) [6] and Timeout Based Purging (TBP), the latter representing a traditional purging scheme. Our experimental results indicate that our hybrid approaches exhibit substantially better performance in important performance metrics: False Positive Rate (FPR), False Negative Rate (FNR) and the number of active flows. One of our hybrid approaches, called "clock"-like Flow Replacement with Lazy Purging (CFR-LP), reduces the number of false positives and false negatives of FCF by 87.1% and 78.4% and Also it outperforms TBP by 67.6% and 13.3% when the table size is twice the average number of active flows. This tendencies were steadily observed whether there is sufficient space or few empty space on the flow table. In particular, this CFR-LP showed the best accuracy of the estimation of the number of active flows. Also, we find that our CFR-LP reduce the FPR and FNR of TBP by 65.6% and 47.6% and FCF by 79% and 36.1% when the table size is twice the average number of active flows. Finally, our CFR-LP can also confine heavy hitters to a paricular state without counting the number of packets. Our experiments showed that the number of flows in the state are less than 10% of active flows and 0.5% error rate on its accuracy.

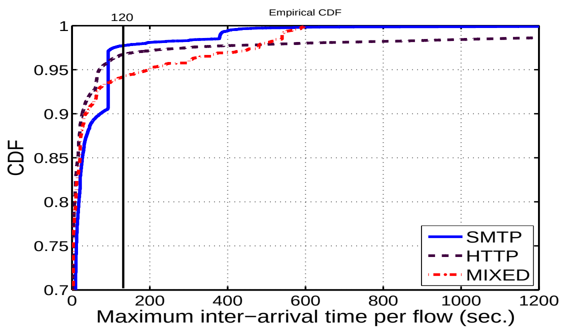


Fig. 1. Maximum inter-packet time per flow

The rest of this paper is organized as follows: In Section II, we discuss background information including previous approaches. In Section III, our "clock"-like replacement algorithm and its variants are described including the hybrid approaches. Then, implementation issues of our schemes are discussed in Section IV. Experimental evaluation comparing those approaches is given in Section IV followed by conclusions and a discussion of future work in Section VI.

II. Previous approaches

In order to provide a concrete overview, we assume there is a per-flow monitoring system for bidirectional traffic in a linecard (observed aggregate packet flow). Specifically, we consider a flow table consisting of $Flow\ ID$, $touch\ bit$, and $state$. To enable per-flow monitoring of high-end routers, the use of fast memory, *e.g.*, SRAM, is required. We simply assume that the flow table is implemented using a hash table with hash-chaining (We do not address the problem of hash collisions in this paper.). Papers have preferred to this problem, *e.g.*, Shared Fast Hash Table [28], FCF [6], or a TCAM-based approach [23]. These approaches can be combined with our replacement frameworks.

A flow ID is composed of several fields of the TCP and IP header, typically including the following 5-tuple [9]:

$$\{src\ ip, dst\ ip, src\ port, dst\ port, ip\ proto\} \qquad (1)$$

In addition, $timestamp$ and $state$ may be supplemented to constitute a 128 bit flow entry, in stateful-inspection firewalls [29]. Note that a hashed flow ID 32 bits may be employed to reduce the flow entry size. However, there is a trade-off between the size of the flow ID and hash collisions [23].

For comparison purposes, TBP, a traditional purging scheme based on touch bits, is described in this section. Currently proposed purging schemes, such as [6], [16], use a touch bit to purge stale flows as a fast implementation of the timeout processing. In these schemes, one bit touch bit is used to check flow's activeness during the measurement time interval T_θ. Incoming packets modify the corresponding flows' touch bit. Thus, at the end of every time interval T_θ, flows whose touch bits were not activated may be purged. This scheme may require the traversal of all flow entries at the end of every timeout interval.

A problem of TBP, as discussed previously, is how to set up the timeout threshold value, without causing flow table explosion or prematurely purging active flows. Figure 1 shows the cumulative distribution function of maximum inter-packet time per flow for three Internet traces: SMTP, HTTP, MIXED [2], [3]. From this figure, we can see the relationship between a timeout threshold and the number of prematurely purged flows. For instance, if a timeout threshold T_θ is 30 seconds, up to 20% of flows may be wrongly purged, *i.e.*, false negatives, while $T_\theta = 120$ prematurely purges around 3-7% of active flows. In current networking devices [8], [27], 3600 seconds is used to expire established TCP connections. Even though this greatly reduces false negatives, it requires a huge-sized flow table. NetFlow [1] uses 15 seconds, while 60 seconds is used in Adaptive NetFlow [10]. These various settings of the timeout threshold implies that choosing a proper timeout value is not straightforward, motivating us to investigate other approaches besides TBP. Calibrating TCP session-level timeouts is explored in [15], [19]. Kim *et al.* [15] demonstrated that the long time-out threshold under SYN flooding results in a session table explosion. They only discussed the problem for the initial stage of a TCP session, but this kind of time-out mechanism experiences such trade-offs throughout the entire session lifetime.

A feasible and efficient management of finite-state machines (FSM) monitoring active flows is discussed in [6], where each FSM (ACSM in [6]) approximates a single TCP session's state. [6] described "bucket"-level purging, rather than a flow-level purging. Similarly, [31] discussed the significant overhead of massive purging and introduced a late purging concept. While a hash chain corresponding to the flow ID of the incoming packet is searched in the flow table (hash table), outdated flows in the hash chain are checked to be purged. Both the above techniques, however, may give rise to high false positives, thereby overestimating the number of active flows.

Congestion control focuses on decreasing the average throughput of the long-lived TCP flows [13]. Congestion-responsive persistent flows and legitimate short-lived flows are obviously both important but may need to be separately managed in packet memory to protect the congestion control mechanism of the former. [12], [20], [25], [24], [32] purpose how to protect against unresponsive TCP flows. Unresponsive TCP flows' occupation rate of buffers in the routers tends to be higher during network congestion periods. To find those flows and regulate them, two kinds of methods, with and without per-flow state, have been examined. Although per-flow state produces accurate information about TCP session connections, implementation issues of concern in this paper have not yet been solved.

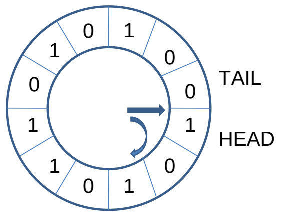


Fig. 2. A visual description of the "clock" algorithm

Estimation methods for the concurrent number of flows in [24], [32], [21] avoid the overhead of keeping per-flow state and, if these estimation methods are proven feasible, they can be applied to estimate combined flow aggregates (with both short-lived flows and long-lived flows) because the overhead of storing per-flow state is reduced. However, when many short-lived TCP flows combine with long-lived TCP flows, estimation becomes "unstable" because the cache hit rate of short-lived flows is relatively lower than that of long-lived ones.

III. CLOCK-LIKE FLOW REPLACEMENT

In this section, we describe a "clock"-like Flow Replacement (CFR) algorithm and present some improvements on it. CFR is stimulated by the "clock" [5], a page replacement algorithm, so that the flow table also can benefit from replacement to avoid flow overflow. The "clock" concept is simple to implement and performs similarly to LRU cache algorithm in terms of hit page replacement, but with much reduced operational complexity for maintaining the ordered list. In "clock", as shown in Figure 2, cached pages are maintained by using a circular buffer, called a $clock$, and each page has a page reference bit (a touch bit in our algorithm) indicating its reference. If a page hit occurs, the page reference bit is set to 1. If a page miss occurs, then an evicted page is selected as follows: look at the current page under the HEAD pointer of the buffer and check its page reference bit. If the page reference bit is 1, set it to 0, move forward in clockwise rotation, and check the page reference bit of a next entry until it is 0. This entry is then selected (a "victim"), and replaced with a new page referenced.

In CFR, the touch bit is not set based on timeout, but depends on the $clock$ lookup. For this CFR, an additional circular list, a $clock$, is needed to maintain the order of replacement. This linked list could be a part of flow entry in the flow table or it could be separate. Since this single list could also be a burden for the SRAM, we will describe a bitmap algorithm without using the $clock$ later. The size of the $clock$ can be fixed as the flow table size M. When a packet arrives, a flow update (and replacement) is performed as follows:

- **If a flow "hits" in the flow table**, update the entry and set the touch bit to 1. If a flow is normally closed by observed a FIN or RST packet, the touch bit is set to 0.
- **If a flow "misses" in the flow table,** and the flow table is full, find a victim entry from the $clock$ (The procedure

State	Touch bits	Flow type
V_0	00	unused
V_1	01	initial state
V_2	10	short-lived
V_3	11	long-lived

TABLE I
TOUCH BIT STATES IN CFR-2

of finding an victim entry is equal to the "clock".). If the table is not full, add a new entry to the $clock$.

One good thing of this CFR algorithm is that the table overflow does not occur even when the flow table becomes full. Since even if the flow table is full and all the entries' touch bits are 1, we still traverse once through the $clock$, resetting all these bits to 0, and finally visit the HEAD entry of the $clock$ again as an evicted flow because its touch bit is 0 now. On the other hand, this CFR may have a significant overhead trying to find a victim entry. Thus, we discuss a bitmap implementation and a provisioning lookup approach later. We also give an overhead analysis showing that only one memory accesses would be sufficient to find a victim entry in most cases.

A. CFR-2

If we extend the touch indicator to two 2 bits, it can contain up to four different states, giving better performance, especially if we need to differentiate among SYN floods, short-lived flows, and long-lived flows. Table I shows the touch bits and the corresponding flow state of CFR-2. In most cases, the relationship between touch bits and flow type follows Table I. However, it may not be true in all the cases. The escalation of the touch bit can be possible according to the frequency of packets as follows:

- **If the flow "misses" in the flow table**, *i.e.,* if the flow ID of an incoming packet does not match an entry in the flow table, find an evicted entry, and replace it with the new flow, setting its touch bit to 01.
- **If the flow "hits",** update the entry and increment its touch bit by 1 with probability p_1 or p_2. Upon receipt of a RST or FIN packet, the touch bit of this flow is reset to 00.

Here, we adopt two probabilities p_1 and p_2 to increment the touch bit of the flow in V_1 and V_2, respectively. For example, if a flow i is new, it starts with its touch bit 01. if $p_1 = 1$ and $p_2 = 0.125$, the second packet of flow i sets its touch bit to 10. On average, one packet among the next eight packets of flow i with $p_2 = 0.125$ makes the flow transits to a state V_3. Since the long-lived flow has more than 10 packets [22], long-lived flows can reach to state V_3.

In this scheme, the touch bits (and its state) can give multi-level priority for the flow removal. Each priority informs the frequency level of the packet flows in this scheme. For example, touch bits 10 (state V_2) on average means this flow has less than 10 packets so far. Touch bits 11 state V_3) indicates that this flow is a long-lived flow on average. By decreasing

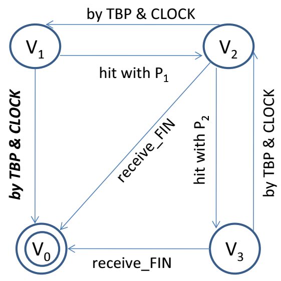


Fig. 3. State transition diagram in CFR-PD

the probabilities p_1 and p_2 lower, we can confine heavy-hitters to the state $V3$ from normal flows. For example, if we have a setting of $p_1 = 0.1$ and a very low value of p_2, we can locate normal long-lived flows in state $V2$ and heavy-hitters in state $V3$. Selecting the probabilities p_1 and p_2 is a engineering issue which we will discuss in context of detecting heavy hitters. Note that in terms of a purging priority, not heavy hitters but long-lived flows have meaningful discrimination from short-lived flows.

When a new flow is inserted into the flow table, its touch bits start from 01, rather then 10, thus protecting the flow table from contamination by SYN flooding packets, because this malicious flows, consisting of a spoofed SYN packet, will be first evicted due to a low level of the touch bit when the flow table gets full. Also, a small amount of state allocation, such as SYN cache [17], can be assigned to the flows in the initial stage of flow connection to mitigate DoS attacks.

When the flow table is full, the "clock" mechanism executes to find an evicted entry from the $clock$. It decrements the touch bit by 1 until it arrives to an unused entry. This movement makes transition to the lower level state, but an active flows can promptly recover its original state. Therefore, this circulation helps to purge the outdated flows.

IV. HYBRID APPROACHES

In order to more accurately estimate the number of active flows, timeout mechanism can be combined with CFR scheme. In this section, we discuss two kinds of hybrid approaches. First, we propose a continuous timeout processing as a combination of CFR-2 and TBP, called a "clock"-like Flow Replacement with Periodic Decrement (CFR-PD). Second, we discuss "lazy" purging to benefit both CFR and TBP.

A. CFR-PD

Figure 3 shows state transition of the touch bit in CFR-PD and Algorithm IV.1 shows a simplified version of CFR-PD, without employing probabilities. In this scheme, the TBP is employed to reduce the touch bits, thereby having a role in purging outdated flows, besides the decrement of the touch bits by the "clock". Thus, we need a timeout threshold t_θ and a node traversal to examine all the flow's activeness by checking touch bits at the end of every t_θ. Unlike the single touch bit of TBP, the touch bits are decremented by 1 unless at 00. This gives one level decrement of flows which were idle for last T_θ. In CFR-PD, TBP gives the same effect of

Algorithm IV.1 A simplified CFR-PD algorithm

```
1: on arrival of packet $p$ of flow $f$: {// “clock” function}
2: if $f \in FlowTable$ then
3:     update $f.state$
4:     if $p$ is FIN or RST then
5:         $f.touch \leftarrow 00_2$ { // 0 in decimal}
6:     else if $f.touch < 11_2$ then { // 3 in decimal}
7:         $f.touch \leftarrow f.touch + 1$
8:     end if
9: else {//$f \notin FlowTable$}
10:     while $HEAD.touch \neq 00_2$ do {// find a victim entry}
11:         $HEAD.touch \leftarrow HEAD.touch - 1$
12:         $HEAD \leftarrow next[HEAD]$
13:     end while
14:     entry $e \leftarrow HEAD$
15:     update $e.state$
16:     $e.touch \leftarrow 01_2$ { // 1 in decimal}
17:     $FlowTable \leftarrow FlowTable \cup \{e\}$
18:     $HEAD \leftarrow next[HEAD]$
19: end if
20:
21: at the end of the timeout threshold $T_\theta$: {// $TBP$ function}
22: for all $e \in FlowTable$ do
23:     if $e.touch \neq 0$ then
24:         $e.touch \leftarrow e.touch - 1$
25:     end if
26: end for
```

circulational decrement (a “clock” activity) in CFR-2. Any decremented active flows can be restored to their state soon, so TBP only affects to the idle flows. From this case, we can see that second (or third) continuous decrement makes idle flows purged, as shown in Figure 3. In case the touch bit was 01, it is decremented to 00, which means a flow is terminated. Also, we can keep these idle flows in the flow table, until there is no available space so as to benefit from the replacement in the flow table. Thus, a couple of periodic decrementations can reduce false negatives. However, CFR-PD may overestmate active flows because continuous decrementations delay purging. Therefore, we need a scheme which makes the estimation of active flows more accurate and propose another hybrid approach which fulfills these requirements.

B. CFR-LP

As discussed in our experiments later, TBP showed a tendency to overestimate the number of active flows. Particularly, our CFR-PD has this feature because it has the same effect of employing multiple timeout thresholds to avoid false negatives.

To improve the accuracy of the estimation of active flows, we suggest “clock”-like Flow Replacement with Lazy Purging (CFR-LP). Figure 4 shows state transition of the touch bits in CFR-LP and Algorithm IV.2 shows TBP portion of CFR-LP, *i.e.*, decrementing the touch bit by 1, which is the only improvement to CFR-PD, is effective only to the flow whose

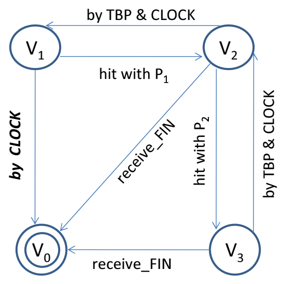


Fig. 4. State transition diagram in CFR-LP

Algorithm IV.2 TBP in CFR-LP algorithm

```
1: at the end of the timeout threshold $T_\theta$: {// $TBP$ function}
2: for all $e \in FlowTable$ do
3:     if $e.touch > 1$ then
4:         $e.touch \leftarrow e.touch - 1$
5:     end if
6: end for
```

touch bits are greater than 01. Also, only if the touch bit is greater than 10, this flow is regarded as active. Thus, if a flow is purged and its touch bit is 01, then it is not counted as an active flow. However, this flow is not actually be purged from the flow table and also is not decremented to 00 by TBP. Thus, this flow can be restored to normal state by just incrementing the touch bit by 1.

While TBP finds candidates to be purged (by setting those touch bits to 01), “clock” mechnasim actually finds and replaces a victim entry among them, hence we call it “lazy” purging. Purged flows in state V_1 can be removed from the flow table by the “clock” activity only when there is no unused entries. The removal of the flow entry is based on multi-level priorities strictly differentiated, *i.e.*, there are two types of inactive flows: normally terminated or purged. Each state indicates the frequency level of the packet flows. Within the same level, the $clock$ performs similar to FIFO queueing.

V. Implementation issues

In this section, we discuss ways to reduce the memory overhead to maintain the circular buffer $clock$. A bitmap implementation is described and then TCAM usage is discussed to improve the lookup speed. Ater that a special-purpose flow memory and confining heavy-hitters is described.

A. A bitmap implementation

Considering the SRAM size limitations, it is better to reduce the flow table size. Instead of using the $clock$, we can implement our scheme using a bitmap algorithm, consisting of an M array of 2 bits. Figure 5 shows a bitmap implementation of CFR algorithms. “clock”-wise circulation can be achieved by simply adopting modulo operation.

1) Provisioning Lookup: With a bitmap algorithm, we can significantly reduce the overhead of finding a victim entry along the clockwise direction. To find a victim entry from the bitmap, we can patch a word from $HEAD$ in a memory access, which has a 16 entries of two bit touch bits. We term

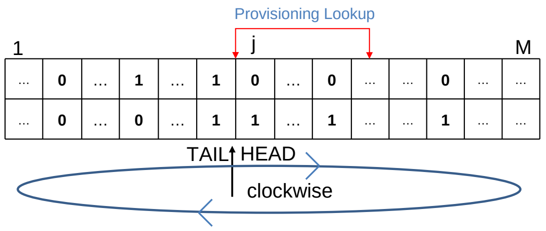


Fig. 5. A bitmap implementation of CFR algorithms

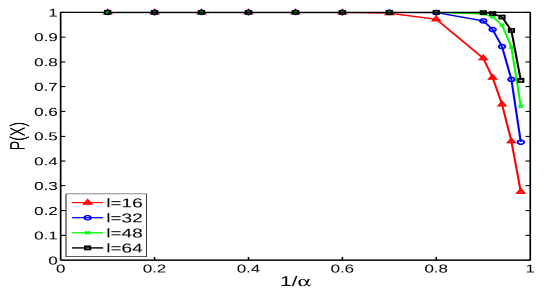


Fig. 6. The lookup overhead in bitmap implementation

this word *provisioning lookup*. Then, we can choose a first unused entry from this.

Preliminary Analysis: We assume that M-sized flow table has three types of entries: active flows in state V_2 and V_3, purged flows in state V_1 and unused entries in state V_1. The number of active flows, purged flows and closed flows are defined a, p and u, respectively. Then, $M = a + p + u$. Let l be the number of entries in a *provisioning lookup*. X is the event that there is at least one unused entry in a *provisioning lookup*.

$$\begin{aligned} P(X) &= 1 - \frac{\binom{a+p}{l}}{\binom{M}{l}} \\ &= 1 - \frac{(M-l)! \cdot (a+p)!}{M! \cdot (a+p-l)!} \end{aligned}$$

Let $a+p = \frac{M}{\alpha}$

$$\begin{aligned} &= 1 - \frac{\frac{M}{\alpha} \cdot (\frac{M}{\alpha} - 1) \cdots (\frac{M}{\alpha} - (l-1))}{M \cdot (M-1) \cdots (M-(l-1))} \\ &= 1 - (\frac{1}{\alpha})^l \cdot \frac{M \cdot (M-\alpha) \cdots (M - \alpha(l-1))}{M \cdot (M-1) \cdots (M-(l-1))} \\ &\geq 1 - (\frac{1}{\alpha})^l, \end{aligned} \qquad (2)$$

where $l \leq \frac{M}{\alpha} \leq M$.

Note that $\frac{1}{\alpha} = \frac{a+p}{M} = \frac{M-u}{M} (\leq 1)$ indicates the load of the flow table. Based on equation (2), Figure 6 shows that relationship between $P(X)$ and $\frac{1}{\alpha}$ as varying l. For example, if we let $\frac{1}{\alpha} = 0.8$, then $u = 0.2M$ *i.e.*, unused entries are 20% of the flow table. In this setting of $\mid l \mid = 16$ and $M = 1,000,000$, $P(X) = 97.19\%$.

Figure 6 convinces that the *provisioning lookup* reduces the "clock" overhead significantly. Considering that the FCF takes at least d, the number of hash functions (usually three or four), times memory accesses to find an empty entry [6], our CFR approach with the *provisioning lookup* requires less memory accesses than previous purging approaches.

B. TCAM implementations of CFR

We can implement the bitmap algorithm on the TCAM whose entry has a flow ID and two bits touch bits.

When a packet arrives, then $lookup(FlowID, *)$, if we let perform $lookup(FlowID, touch\ bit)$, If the lookup hits, updates the entry and increments the touch bit. If the lookup fails, perform $lookup(*, 00)$ to find an evicted entry. If this lookup succeeds, replace this entry with the new one and then decrement the touch bits of flows between $HEAD$ and this lookup result. The new HEAD become the next entry of this lookup result. If this lookup also fails, do $lookup(*, 10)$. $HEAD$ points to the next available entry in the TCAM. However, this TCAM implementation is slightly different from the original algorithm because in our approach the $lookup$ operation returns the first matching entry in case of multiple hits.

To implement our CFR bitmap algorithm more accurately, we need to assume a search operation $lookup(FlowID, touch\ bit, HEAD)$, which returns the first matching address of the Flow ID, searching from the current address of $HEAD$ in the clockwise rotation. If TCAM hits, update the flow entry and set the touch bit correspondingly. Otherwise, perform $lookup(*, 00, HEAD)$ to find an evicted entry without a bit-by-bit visit. If a victim is selected, then set the touch bit of flows between $HEAD$ and a lookup result. If this lookup fails, do $lookup(*, 01, HEAD)$ next by incrementing touch bits. Without the flow ID hashing, the total size of TCAM used is $106 * M$ bits (With the flow ID hashing to 32 bits, $34 * M$ bits are possible.).

C. A special purpose flow table

Besides general purpose per-flow monitoring, there are several special purposes of the flow table for small flow memory. For example, [11] used a flow memory to detect heavy hitters. [16] also mentioned a small flow table for per-flow fair queueing. In these papers, heavy hitters are monitored by a small flow memory in the router, so the flow replacement frequently occurs. However, neither papers describe how to select a victim entry and replace it. It is because LRU has significant overhead of maintaining an ordered list for every incoming packets. Besides the Flow ID lookup in the flow table, a search and its update, *i.e.*, moving the hit entry to the tail (MRU position) should be performed in the ordered list.

On the other hand, our hybrid approaches (especially CFR-PD) has less overhead for maintaining the *clock* list. These hybrid approaches depend on the probabilities p_1 and p_2, so it may lose to select the least recently used one, but it will give a sufficiently precise approximation with a simple implementation. List lookups could be faster by using the bitmap implementation described in the next section. Therefore, our hybrid approaches (CFR-PD and CFR-DP) are a reasonable choice for this type of a special purpose flow memory.

Test result	Actual condition (flow existence)	
	Present (flow existence)	Absent (flow removal)
Positive (Test shows "existence")	True positive (active flows)	False positive (purged flows at next TΦ)
Negative (Test shows "removal")	False negative (**prematurely purged flows**)	True negative (purged or closed flows)

TABLE II
RELATIONSHIPS IN BINARY CLASSIFICATION

D. *Confining heavy-hitters*

By dynamically adjusting the probability p_1 and p_2 according to the varying traffic mix, we can confine heavy-hitters to state V_3. If we let P_θ and N_θ be the total number of packets and the estimated number of active flows during previous T_θ respectively, the fair use of an individual flow will be $\frac{P_\theta}{N_\theta}$. Thus, we can define a heavy hitter TCP flow h such that $h > n\frac{P_\theta}{N_\theta}$, where n is a configuration parameter. Then, we can confine heavy hitters to state $V3$ by adjusing p_1 and p_2, using the following equation:

$$p_1 \times p_2 = \frac{N_\theta}{n \times P_\theta}, \tag{3}$$

For example, If $P_\theta = 10,000$ and $N_\theta = 100$, then the average number of packets per flow is 100. If we define $n = 5$, then $p_1 * p_2 = 0.05$. If we have a simple setting of p_1, say $p_1 = 1$, *i.e.*, any packet of an flow updates its state in V_1 to V_2, then p_2 can be set to 0.05. P_θ and N_θ can be recorded for each time interval and input to the next phase's adjustment of the probabilities p_1 and p_2.

Our hybrid schemes are useful to confine heavy hitters because the touch bit state depends on both the previous phase and current incoming packets. Note that this confinement benefits from the real-time monitoring (without sampling), yet just can give an approximation based on the probabilities *i.e.*, it cannot gaurantee accurate detection of heavy-hitters. However, our framework can reduce the overhead of more accurate monitoring in the flow memory because these full monitoring is only required for the flows of V_3. Hence, it can be applied for congestion control and fair queueing. For instance, confined heavy-hitters are scheduled by using DRR in flow-aware networking [16]. Also state-dependent update, *e.g.*, only one update among ten packets when the flow is a heavy hitter, might be considered to reduce the monitoring overhead itself. Or, it can work as a front-end filter, combined with more accurate counting methods, such as [11], and probability-based detections [24], [32] and [21].

VI. EXPERIMENTAL EVALUATION

A. *Comparison Metrics*

We use three objective metrics: False Positive Rate (FPR), False Negative Rate (FNR) and the number of active flows, to analyze our CFR algorithms and compare them with TBP.

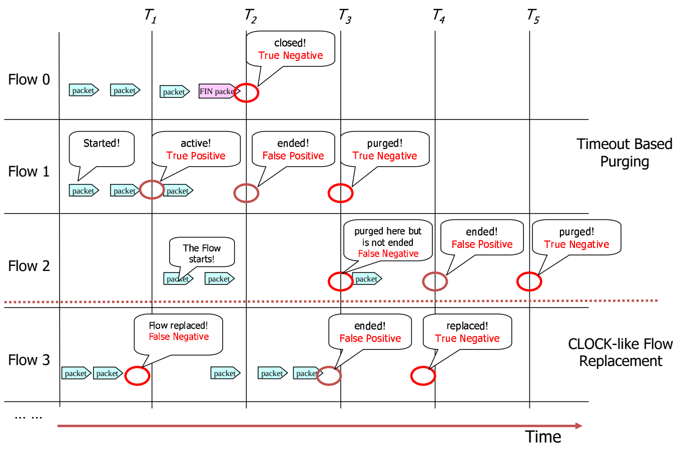


Fig. 7. False positives and false negatives on session purging phase

1) FPR and FNR: False positives, in the context of our per-flow monitoring, indicates that flows that could have been deleted are not deleted, while false negative means flows that should not be deleted are deleted, as shown in Figure II.

One subtle thing of false negative on TBP is the case when there is no available entry for new flows. In this case, this new flow is dropped in our experiments and, as a result, a false negative is added. Therefore, in TBP, when the number of active flows is larger than the flow table size, an overrun of false negatives occurs.

we can observe that

$$FPR = \frac{False\ positive}{False\ positive + True\ negative}, \tag{4}$$

$$FNR = \frac{False\ negative}{True\ Positive + False\ negative}. \tag{5}$$

2) Estimation of active flows: To measure the number of active flows more accurately, a flow's activeness can be reviewed in terms of TBP. Some portion of idle flows might be simply idle for a longer interval than a timeout threshold T_θ, while others are unused further so as to be purged through TBP. For instance, when flow 0 ends with a FIN packet as shown in Figure 7, the flow is deleted from the flow table at the correct time and is regarded as a true negative. However, if flow 1 ends with a time-out, this flow is purged at T_3 as the flow has not been touched in the next T_ϕ. In fact, this flow could have been deleted at T_2, so it is a false positive at T_2 and a true negative at T_3. More complex example is flow 2, whose packets are incoming with a larger time interval than the given time-out threshold T_ϕ, the flow is purged. However, if there is another packet of flow 2 coming in just past T_3, a false positive and a true negative which have already been counted as in flow 1 should be invalidated, and then flow 2 ends with a new false positive at T_4 and a true negative at T_5. Similarly for CFRs, when flow 3 is replaced before T_1, even though an additional packet is pending, this is a false negative. When flow 3 ends before T_3 and is replaced before T_4, it results in a false positive at T_3 and a true negative at T_4.

Metric	False positives			False negatives		
Table size	$0.5X$	$2X$	$5X$	$0.5X$	$2X$	$5X$
CFR	0	2799.8	12314.4	1446.2	418.9	277.9
CFR-2	0	4549.1	16291.1	1262.5	360.2	248.3
TBP	1835.4	3535.9	3535.9	8897.8	452.3	452.3
FCF	2742.2	10099.6	8926.5	11054.2	1820.1	368.6
CFR-PD	0	2303.7	908.2	1475.6	632.7	527.9
CFR-LP	0	1659.2	1143.7	1272.6	392.1	273.7

TABLE III
THE COMPARISON OF FLOW MANAGEMENT ALGORITHMS: FALSE POSITIVES AND FALSE NEGATIVES (HTTP TRACE, $T_\theta = 60$)

Table size	$0.5X$	$2X$	$5X$
N_a	2838.8	6179.7	6179.7
TBP	4425	8850.9	8850.9
FCF	4158.7	14756.66	14065.0
CFR-PD	3016.0	7952.7	9324.1
CFR-LP	2904.0	6683.2	6568.3

TABLE IV
THE COMPARISON OF FLOW MANAGEMENT ALGORITHMS: THE AVERAGE NUMBER OF ACTIVE FLOWS (HTTP TRACE, $T_\theta = 60$)

From Fiqure 7, we can drive an equation to estimate the number of active flows (called N_a) as follows:

$$N_a(T_\theta) = TP(T_\theta) + FN(T_\theta) - (FP(T_\theta) - FN(T_{\theta+1})). \quad (6)$$

B. Experimental Setup

Experiments were done using real Internet traces captured by LBL [3] and DATCAT [2] to investigate practical results for comparison among the proposed approaches. The traces of SMTP, HTTP, and MIXED were selected as examples of short-lived flows, long-lived flows, and mixed flows respectively. The SMTP trace, which was captured for 61 minutes, consists of 36,099 total flows including 404-576 concurrent flows. The HTTP trace, also captured for 61 minutes, has 413,371 flows with a maximum of 6,035 active flows simultaneously. The MIXED trace of 10 minutes duration is composed of 2,491 total flows with a maximum of 275 concurrent flows including HTTP, SMTP, LDAP and any other ports. The time-out threshold T_ϕ was varied as 5, 30, 60, and 120 seconds. We used $p_1 = 1$, $p_2 = 0.1$ for the hybrid approaches.

C. Experimental Results

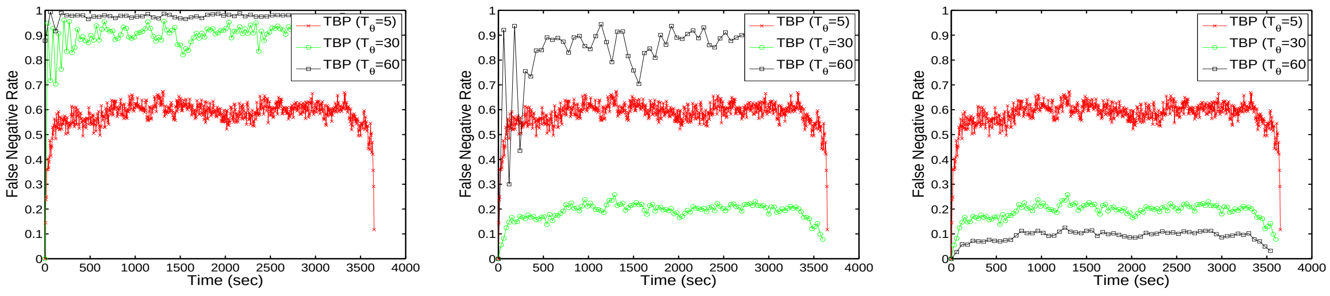
1) Varying T_ϕ: Figure 8 shows FNR at TBP of the SMTP trace with varying T_ϕ (5, 30 and 60 seconds). The size of flow table varies as 500, 1,000, and 5,000 because the average number of active flows with $T_\theta = 30$ was about 500. We can see that 5 seconds was too short for the time-out threshold to store active flows. Although the larger T_ϕ allows more flows to be presented in the flow table, we can see the flow table overflows with $T_\phi = 30, 60$ in Figure 8(a) and $T_\phi = 60$ in Figure 8(b). Even when the flow table size is sufficient for active flows, this problem can occur with network attacks such as SYN flooding or scanning [15]. Also, Figure 8(c) shows that TBP with $T_\theta = 30$ misses around 20% of normal flows. This behavior is not due to flow table size, but because of the time-out threshold. Therefore, the experiment results show that a certain amount of FNR, however small, is present in TBP.

2) False positives and false negatives: To observe the relationship between flow table size and false positves (and flase negatives), we conduct the experiments for three traces as varing the flow table $0.5X, 2X, 5X$. where X is the average number of active flows in TBP, measured from Figure 8. T_ϕ is set to 60 seconds. Table III shows the average number of false positives and false negatives for the HTTP trace. From the table, we can see that false negatives and false positives in CFR and CFR-2 are inversely proportional, while both false positives and false negatives in CFR-PD and CFR-LP are decreasing conversely as the flow table increases. FCF shows the largest number of false positives due to bucket-level purging. It only reasonable result of false negatives when the flow table has enourmous space *i.e.,* $5X$. Overally, our CFR approaches quite accurate irrespective of the dynamics of the traffic mix. CFP-LP was the best performer in terms of false positives and false negatives.

3) The estimation of active flows: Figure IV shows the estimation of active flows using equation (6), TBP, FCF, CFR-PD and CFR-LP varying the table size. In the N_a, actual number of active flows are all the same, but active flows are computed as 2838.8 only for the active flows where the flow table contain in case of $0.5X$.

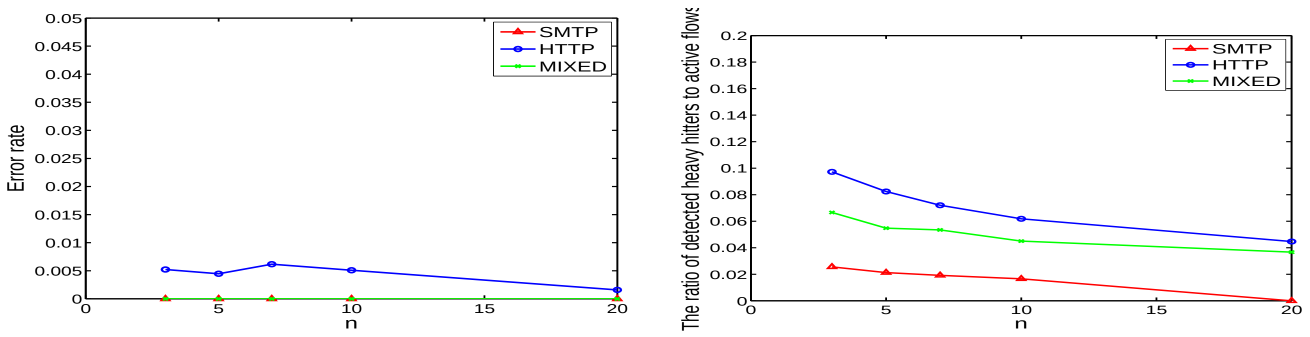
Compared with TBP, CFR-PD overestimates the number of active flows as the flow table size is increased because continuous timeouts only make idle flows purged. On the other hand, CFR-LP underestimates the number of active flows because idle flows are purged by both the purging mechanism (TBP) and the “clock”. Considering the fact that TBP shows an overestimated tendency, as described previously, we can see that CFR-LP is the closest to the accurate measurements (N_a). Note that CFR and CFR-2 cannot be used to measure the active flows because they stores the flows in the flow table as many as the flow table has available space. Also, the experiment showed that FCF could not be used to estimate the active flows.

4) Confinement of heavy hitters: To see how our framework can confine heavy hitters in the higest state (call state $V3$), we measured error rate and the ratio of flows in $V3$ to active flows as shown in Figure 9. We used parameters of $T_\theta = 60$ and $p_1 = 0.1$. p_2 is computed as described in equation 3. Figure 9 shows that heavy hitter candidates, *i.e.,* flows in state $V3$ are below 10% of active flows and the error rate of heavy hitters is quite small (less than 0.5%). This implies that we do not need to monitor all the flows, but we can monitor only these flows in state $V3$ to detect heavy hitters accurately. To monitor these flows packet-by-packet, any detection mechanism such as multi-state filter [11] can be combined. Therefore, our framework can confine heavy hitters in state $V3$ and help to reduce the overhead of detecting heavy hitters by reducing the number of flows to be observed.



(a) Table size=500 (b) Table size=1,000 (c) Table size=5,000

Fig. 8. FNR of TBP varying flow table size



(a) Error rate (b) The ratio of heavy hitters in $V3$ to active flows

Fig. 9. Heavy hitter confinement

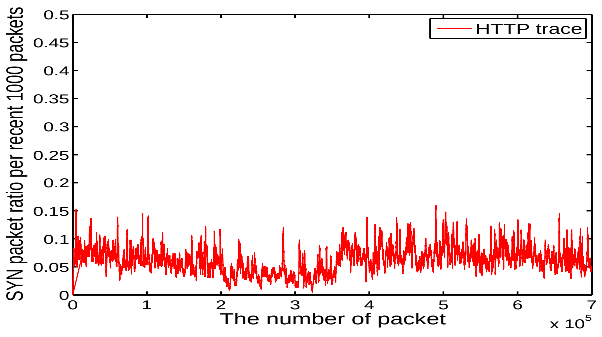


Fig. 10. Monitoring SYN packet ratio

5) Performance under SYN flooding: In general, SYN flooding attacks can be detected and mitigated by front-end intrusion detection and response mechanisms, but there is a possibility that a limited undetected SYN flood can impact a per-flow monitor. One simple technique to detect SYN floods is to monitor the SYN rate. From Figure 10, we can see that the SYN rate over a 1000 packet sliding window in the HTTP trace varied between 1% and 15% with a mean of 6.38%, (the SMTP trace was similar). So, an IDS could be conservatively calibrated to issue a SYN flood alert when the observed SYN rate exceeds 18% over consecutive windows. Thus, for the nominal trace of Figure 10, an injected 3% flood would go undetected, whereas a 10% flood would quickly trigger an alert. Thus, for a 3% SYN flood, we want to show the robustness of the per-flow monitoring system. To achieve, spoofed SYN packets, each of which is an individual flow, were randomly generated and continuously injected among normal packets, after being read from the HTTP trace.

Note that non-threshold methods, *e.g.*, SYN cache [17] have the potential to effectively detect or eliminate SYN floods, but these methods do not resolve the fundamental problem of bandwidth exhaustion created as a result of SYN flood. Also, these methods are undermined if the SYN-ACK does not follow the same path as that of the SYN packet. Moreover, these mechanisms themselves should be resilient to SYN floods, else normal flows cannot be instantiated. Finally, they may react with a “false positive” when encountering non-SYN packets of a flow that was dynamically routed.

To see how to manage normal flows under SYN flooding in our approaches, 3 % of spoofed SYN packets, which form individual flows each, are continuously injected among normal packets read from a SMTP trace. Under SYN flooding, FPR and FNR for the normal flows only are examined. As shown in Table V, CFR-LP still performs better under the SYN flooding. FCF showed low FNR only when the flow table is conceivably large. Note that actually 2.61% of FNR with $5X$ comes from the exaggerating of the estimation of active flows, as shown previously. Table V shows that 3% of injected SYN packets increase over twice the number of active flows, which is around 1150. This increase may cause the overflow of the flow table when the memory resources is limited in the router.

Metric	FPR (%)			FNR (%)		
Table size	$1X$	$2X$	$3X$	$1X$	$2X$	$3X$
TBP	24.80	33.69	35.05	55.13	18.00	6.72
FCF	52.11	55.15	56.81	42.94	14.78	2.61
CFR-PD	0.01	4.59	6.98	32.26	17.80	11.86
CFR-LP	0.17	11.58	30.96	26.98	9.44	3.93

TABLE V
THE COMPARISON OF FLOW MANAGEMENT ALGORITHMS UNDER 3% SYN FLOODING (SMTP TRACE, $T_\theta = 30$)

To avoid this, the flow table size should be as sufficiently large (at least three times) so as not to miss the active normal flows. However, this enlargement of the flow table make it costly, especially when using the fast SRAM memory in the high-end router. Otherwise, SYN flooding affects the routers doing packet inspection by adding a large number of abnormal flows and deleting those flows simultaneously in TBP resulting in the degradation of its performance and eventually downgrading the network connected to it. An alternative is that those abnormal packets doing harm should be detected before reaching to the routers.

VII. Conclusions and Future Work

In this paper, we have proposed a new kind of flow table implementation, based on the simple and efficient replacement policy "clock", and hybrid approaches combining both purging and replacement. Benefits come from multiple states allowing broad clustering of flows triggered by incoming packets, purging and replacement. Theoretical and simulation studies have shown that our frameworks can effectively maintain session state of persistent TCP flows regardless of flow memory size and mallicious attacks. Our approaches can accommodate limited hardware and software resources in addition to very high data rates by avoiding flow table overflow and maintaining an ordered list. Also, Our hybrid approaches can adapt suitably from general purpose to special purpose per-flow monitoring system. Thus, the proposed approaches are effectively applicable for a high-speed networking devices in terms of reduced hardware/software complexity and enhanced real-time processing.

We have pending performance results ow to quickly and efficiently migrate and merge flow based data among our frameworks. Based on the framework to reduce the volume of reporting data by clustering the flows according to their frequency. we will design a resilient and collaborative IDSes within an enterprise level in the future. A major concern is that such infomaion sharing could be maliciously induced as part of a defense-softening pre-attack. Thus, it should be carefully considered to detect anomalious high volume of such information sharing.

References

[1] Cisco NetFlow. http://www.cisco.com/warp/public/732/Tech/netflow.
[2] Internet Measurement Data Catalog. http://www.datcat.org/.
[3] Lbnl/icsi enterprise tracing project. http://www.icir.org/enterprise-tracing/.
[4] Comcast's broadband usage cap won't hurt my mom. http://www.pcmag.com/article2/0,2817,2331596,00.asp, October 2008.
[5] Sorav Bansal and Dharmendra S. Modha. CAR: Clock with Adaptive Replacement. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 187–200, Berkeley, CA, USA, 2004. USENIX Association.
[6] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. Beyond bloom filters: from approximate membership checks to approximate state machines. In *Proceedings of ACM SIGCOMM*, volume 36, pages 315–326, 2006.
[7] Check Point Software Technologies, Ltd. FW1. http://www.checkpoint.com/.
[8] Cisco Systems, Inc. IOS XR 12000 Router. http://www.cisco.com/en/US/products/ps6342/index.html.
[9] Nick Duffield, Carsten Lund, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of ACM SIGCOMM*, pages 325–336, 2003.
[10] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better NetFlow. In *Proceedings of ACM SIGCOMM*, pages 245–256, 2004.
[11] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM*, pages 323–336, 2002.
[12] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.
[13] Liang Guo and Ibrahim Matta. The War between Mice and Elephants. In *Proceedings of IEEE ICNP*, page 180, 2001.
[14] ISS. SiteProtector. http://www-935.ibm.com/services/us/index.wss/offering/iss/a1027232.
[15] Hyogon Kim, Jin-Ho Kim, Inhye Kang, and Saewoong Bahk. Preventing Session Table Explosion in Packet Inspection Computers. *IEEE Trans. Computers*, 54(2):238–240, 2005.
[16] Abdesselem Kortebi, Luca Muscariello, Sara Oueslati, and James Roberts. Minimizing the overhead in implementing flow-aware networking. In *ANCS '05: Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*, pages 153–162, New York, NY, USA, 2005. ACM.
[17] Jonathan Lemon. Resisting SYN flood DoS attacks with a SYN cache. In *BSDC'02: Proceedings of the BSD Conference 2002 on BSD Conference*, 2002.
[18] Kirill Levchenko, Ramamohan Paturi, and George Varghese. On the difficulty of scalably detecting network attacks. In *Proceedings of ACM CCS*, pages 12–20, 2004.
[19] Xin Li, Zhenzhou Ji, and Mingzeng Hu. Session Table Architecture for Defending SYN Flood Attack. In *Proceedings of ICICS*, pages 220–230, 2005.
[20] Dong Lin and Robert Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM*, pages 127–137, 1997.
[21] Yi Lu, Balaji Prabhakar, and Flavio Bonomi. ElephantTrap: A low cost device for identifying large flows. In *Proceedings of High-Performance Interconnects 2007*, pages 99–108, 2007.
[22] Marco, Ion Stoica, and Hui Zhang. TCP model for short lived flows. *IEEE Communications Letters*, 6(2):85–87, 2002.
[23] Gunwoo Nam, Pushkar D. Patankar, George Kesidis, and Chita R. Das. Managing per-flow state of TCP sessions in Internet routers: Session Purging. The Pennsylvania State University Technical Report CSE08-120, 2008.
[24] Teunis J. Ott, T. V. Lakshman, and Larry H. Wong. SRED: Stabilized RED. In *Proceedings of IEEE INFOCOM*, pages 1346–1355, 1999.
[25] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis. CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *Proceedings of IEEE INFOCOM*, pages 942–951, 2000.
[26] Vern Paxson. Bro: a System for Detecting Network Intruders in Real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.
[27] Snort. Snort Intrusion Detection and Prevention system. http://www.snort.org/.
[28] Haoyu Song, Sarang Dharmapurikar, Jonathan Turner, and John Lockwood. Fast Hash Table Lookup Using Extended Bloom Filter: An Aid

to Network Processing. In *Proceedings of ACM SIGCOMM*, pages 181–192, 2005.

[29] Lance Spitzner. Understanding the fw-1 state table. http://www.spitzner.net/fwtable.html.

[30] Symantec. ManHunt. http://www.symantec.com/.

[31] Jun Xu and Mukesh Singhal. Cost-Effective Flow Table Designs for High-Speed Routers: Architecture and Performance Evaluation. *IEEE Trans. Comput.*, 51(9):1089–1099, 2002.

[32] Sungwon Yi, Xidong Deng, George Kesidis, and Chita R. Das. A Dynamic Quarantine Scheme for Controlling Unresponsive TCP sessions. *Telecommunication Systems*, 37(4):169–189, 2008.

[33] Seungyong Yoon, Byoung koo Kim, Jintae Oh, and Jongsoo Jang. High Performance Session State Management Scheme for Stateful Packet Inspection. In *APNOMS*, pages 591–594, 2007.