

Technique for Estimating the Number of Active Flows in High-Speed Networks

Sungwon Yi, Xidong Deng, George Kesidis, and Chita R. Das

The online collection of coarse-grained traffic information, such as the total number of flows, is gaining in importance due to a wide range of applications, such as congestion control and network security. In this paper, we focus on an active queue management scheme called SRED since it estimates the number of active flows and uses the quantity to indicate the level of congestion. However, SRED has several limitations, such as instability in estimating the number of active flows and underestimation of active flows in the presence of non-responsive traffic. We present a Markov model to examine the capability of SRED in estimating the number of flows. We show how the SRED cache hit rate can be used to quantify the number of active flows. We then propose a modified SRED scheme, called hash-based two-level caching (HaTCh), which uses hashing and a two-level caching mechanism to accurately estimate the number of active flows under various workloads. Simulation results indicate that the proposed scheme provides a more accurate estimation of the number of active flows than SRED, stabilizes the estimation with respect to workload fluctuations, and prevents performance degradation by efficiently isolating non-responsive flows.

Keywords: Flow estimation, Markov model, non-responsive flows, SRED, HaTCh.

Manuscript received July 25, 2007; revised Oct. 18, 2007.

A preliminary version of the paper was presented at IEEE CDC, Dec. 2003.

Sungwon Yi (phone: +82 42 860 4865, email: sungyi@etri.re.kr) is with the S/W & Content Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Xidong Deng (email: xdeng@stcloudstate.edu) is with the Department of Electrical and Computer Engineering, St. Cloud State University, Minnesota, USA.

George Kesidis (email: kesidis@cse.psu.edu) and Chita R. Das (email: das@cse.psu.edu) are with the Department of Computer Science and Engineering, Pennsylvania State University, Pennsylvania, USA.

I. Introduction

Measuring and monitoring traffic is an important but admittedly difficult problem, primarily because of the huge volume of traffic to be processed in high-speed networks. To accurately capture the characteristics of network traffic, measuring devices have to maintain per-flow information. However, the complexity of these operations has been the main obstacle for deploying such devices in high-speed networks. To address this problem, various techniques have been proposed. For example, a sampling technique was standardized by the IETF Internet Protocol Flow Information Export (IPFIX) working group [1], and a variation focused on implementation issues was investigated in [2]. Recently, a variation of a hashing scheme called a space-code bloom filter (SCBF) [3] was proposed to avoid the overhead of per-flow maintenance. An SCBF collects network traffic on the arrival of each packet and periodically stores it in permanent storage devices. The required network information can then be obtained offline. Therefore, SCBF can support fine-grained traffic information, which is essential for network management, planning, accounting, and billing.

On the other hand, obtaining coarse-grained traffic information, such as the total number of active flows, online is important for congestion control and network security [4]-[7]. In this context, flow counting techniques, called direct bitmap and multi-resolution bitmap, have been proposed by Estan and others in [7]. These techniques are based on a *bitmap* data structure, in which each source is hashed into a bit, and a bit is marked when the source is active. In practice, the entire bitmap is divided into blocks, and each block represents a different scale of addresses. The number of set bits, which is weighted by scaling factors, is used to estimate the number of active flows. In [6], a multi-resolution bitmap was extended to a scaled bitmap. The main advantage of

bitmap-based approaches is that the number of flows can be estimated by simple operations with a limited amount of memory. However, it is not clear how it would perform under asymmetric traffic assumptions of today's Internet traffic [8].

Ott and others proposed another class of technique called stabilized RED (SRED) [4]¹⁾. SRED has drawn wide attention because it proposes an on-line technique for estimating the number of active flows, and uses it to respond to network congestion [10]-[12]. In SRED, a small cache memory, called the *zombie list*, is used to record the M most recently seen flows. Each cache line (*zombie*) contains the source and destination address pair, last arrival time, and hit count of the flow. Each arriving packet (source and destination address) is compared with a randomly selected cache line. If the addresses match (a hit), the hit count of the cache line is increased by 1. Otherwise (a miss), the selected cache line is replaced by the arriving flow's address with a replacement probability r . To estimate the number of active flows, SRED maintains a hit frequency $f(t)$ and updates $f(t)$ with $(1-\alpha)f(t-1)+\alpha$ in the case of a hit, and with $(1-\alpha)f(t-1)$ in the case of a miss, where α is a time constant. The inverse of the hit frequency ($f(t)^{-1}$) is used as the estimation of the number of active flows.

Although the idea of SRED, estimating the number of flows without maintenance of the per-flow state, is quite novel, a detailed performance analysis showed several limitations. For example, the estimated number of flows fluctuates as the number of flows increases in the network, which implies that the severity of the congestion is not accurately captured. Unstable estimation results from the inaccurate cache comparison process, that is, random comparisons for a large number of flows. Second, although non-responsive flows such as UDP can be identified by computing the hit count and calculating the *total occurrence* of the flow as described in [4], SRED still underestimates the number of active flows when the traffic mix includes both TCP and aggressive UDP connections. Here, underestimation results from unfair sharing of cache lines. Unlike TCP flows, non-responsive flows do not naturally adapt their sending rates according to the network condition, and thereby aggressively occupy cache lines with SRED. This unfair sharing of cache lines causes more cache hits resulting in underestimation.

To address these problems, we present a mathematical model to analyze the estimation capability of SRED and demonstrate how the steady-state hit frequency of the cache model can be used to estimate the number of active flows. We then propose a modified SRED scheme, called hash-based two-level caching (HaTCh), which uses hashing and a two-level caching mechanism to accurately estimate the number of active flows under various workloads. Unlike the original SRED, in which the entire cache is

a single block, the proposed scheme divides the cache into a fixed number of sub-blocks. With the hashing scheme, each arriving packet is hashed into one of the partitioned subcaches, and the hit frequency is maintained for each subcache. Due to the reduced size of the subcache and the number of flows per subcache, the hit probability of an arriving packet is improved. The hashing scheme stabilizes the estimation through this improved hit probability.

The proposed two-level caching scheme, consisting of a smaller level 1 (L1) cache and a larger level 2 (L2) cache, is similar to the general two-level cache used in processor architecture design. It implies that an L1 cache miss results in an access to the L2 cache. However, the cache inclusion property is not satisfied here. An arriving packet is first compared with a randomly selected L1 cache line. If the addresses match, the hit count is incremented; otherwise, a randomly selected L2 cache line is compared with the packet ID (the source and destination addresses). A hit in the L2 cache results in the cache line being brought to the corresponding cache line in the L1 cache. If the addresses do not match in the L2 cache (a miss), the L2 cache line is replaced with the packet ID with a given replacement probability. The purpose of the L1 cache in HaTCh is to isolate the non-responsive flows from the L2 cache. The two-level caching scheme accurately estimates the number of active flows by isolating the non-responsive flows to prevent monopolization of the L2 cache and to yield more room in the L2 cache for the conforming flows.

We extend the SRED analytical model for the proposed two-level caching scheme to demonstrate its effectiveness in isolating the non-responsive flows. We then analyze the simulations NS-2 performance of the HaTCh scheme through extensive using the simulator [9]. Estimation accuracy and stability of estimation in the presence of non-responsive flows are used as the main performance metrics to compare the proposed scheme with SRED. The simulation results indicate that the two-level scheme stabilizes

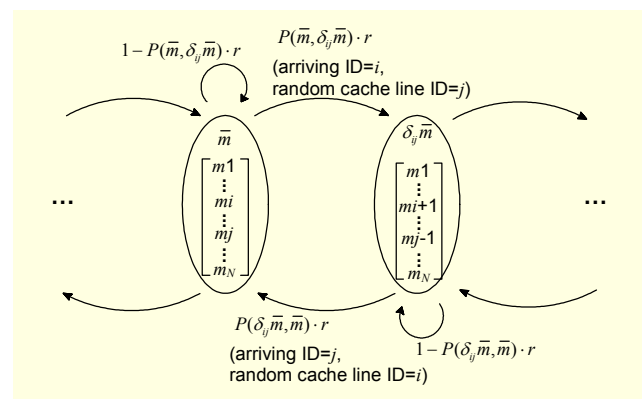


Fig. 1. State diagram of \bar{X} . In this figure, \bar{m} and $\delta_{ij}\bar{m}$ represent the present and the next states. $P(\bar{m}, \delta_{ij}\bar{m})$ denotes the cache miss probability when a flow ID i is compared with a cache line j , and r is the replacement probability.

¹⁾ In this paper, when we mention SRED, we consider only the flow estimation capability of SRED.

the estimation and improves the accuracy of estimation for various workloads. In particular, HaTCh out-performs SRED when the traffic included non-responsive flows. The rest of this paper is organized as follows. In section II, we present a mathematical model to analyze the estimation capability of SRED. We describe the limitations of SRED in section III. The proposed scheme is detailed in section IV. In section V, the simulation results are presented followed by the concluding remarks in section VI.

II. SRED Model for Estimating the Active Flow Number

The key idea of SRED is to relate the cache hit frequency to the number of active flows. However, this was not formally demonstrated in the original paper [3], which relied on simulation to arrive at the conclusion. In this section, we present a Markov model to understand the concept of SRED and analyze its estimating behavior.

Consider a small cache memory, or zombie list, with M lines, and N independent flows, each with a rate of λ_i packets/s. Here, the packet arrival process may *not* be necessarily Poisson. We only assume that the flow IDs of the multiplexed sequence of arriving packets are independent. If we assume X_i is the number of cache lines with the flow ID i in the cache and \bar{X} is the vector that maintains the number of cache lines occupied by each flow, then \bar{X} can be presented as a Markov chain whose transitions occur at packet arrival time \bar{X} , and its state space can be defined as

$$\sum_{i=1}^N X_i = M, \text{ and}$$

$$\bar{X} \in \left\{ \bar{m} = (m_1 \cdots m_i \cdots m_N)^T \mid 0 \leq m_i \leq M, \sum_{i=1}^N m_i = M \right\} \equiv S_{M,N}.$$

Here, the transition rates of the Markov chain are dependent on both the replacement probability and the outcome of the cache comparison between an arriving packet and a randomly selected cache line. Based on SRED's functionality, for a given cache state \bar{m} , the number of cache lines for each flow in the cache remains the same either in the event of a cache hit, or in the event of a cache miss with probability $1-r$. However, the state of the cache changes from \bar{m} to $\delta_{ij}\bar{m}$ in the event of a cache miss with replacement probability r . A pictorial view of the state space transition is given in Fig. 1. When an arriving packet has a flow ID i , and a randomly chosen cache line belonging to flow ID j in the given cache state \bar{m} is replaced by ID i with the replacement probability of r , the new state $\delta_{ij}\bar{m}$ is defined as

$$\delta_{ij}\bar{m} = (m_1 \cdots m_i + 1 \cdots m_j - 1 \cdots m_N)^T \quad \text{if } i < j, \quad (1)$$

for all the cache states \bar{m} such that $m_i < M$ and $0 < m_j$. If $i > j$, then the i and j terms are swapped. For a given state of \bar{m} , the

probability of a cache hit is the product of the probability that an arriving packet has the flow ID i and the probability that a randomly selected cache line has the same flow ID. Similarly, the probability of a cache miss is the product of the probability that an arriving packet has the flow ID i and the probability that a randomly selected cache line has a different flow ID than i . Therefore, the transition probability of this cache model can be written for a cache hit or a cache miss without replacement as

$$\begin{aligned} P(\bar{X}(t+1) = \bar{m} \mid \bar{X}(t) = \bar{m}) \\ &\equiv P_{hit}(\bar{m}, \bar{m}) + P_{miss}(\bar{m}, \bar{m})(1-r) \\ &= \sum_{i=1}^N \frac{m_i}{M} \cdot \frac{\lambda_i}{\sum_{k=1}^N \lambda_k} + \sum_{\substack{i,j=1 \\ i \neq j}}^N \frac{m_j}{M} \cdot \frac{\lambda_i}{\sum_{k=1}^N \lambda_k} (1-r). \end{aligned} \quad (2)$$

For a cache miss that is replaced with probability r , the expression is

$$\begin{aligned} P(\bar{X}(t+1) = \delta_{ij}\bar{m} \mid \bar{X}(t) = \bar{m}) \\ &\equiv P(\bar{m}, \delta_{ij}\bar{m})r = \sum_{\substack{i,j=1 \\ i \neq j}}^N \frac{m_j}{M} \cdot \frac{\lambda_i}{\sum_{k=1}^N \lambda_k} r. \end{aligned} \quad (3)$$

In the above expressions, $\lambda_i / \sum_{k=1}^N \lambda_k$ denotes the probability that an arriving packet has a flow ID i , and the m_i/M and m_j/M terms represent the probability that a randomly selected cache line from the zombie list has the same and different IDs, respectively. The steady-state hit frequency of the given cache model can be expressed by the following theorem.

Theorem 1. Under the assumption of independent packet flow identifiers, the hit frequency, calculated by a first-order autoregressive process for the single cache system (SRED), converges in the steady-state²⁾ to

$$H = \sum_{\bar{m} \in S_{M,N}} \left(\frac{\prod_{n=1}^N \frac{\lambda_n^{m_n}}{m_n!}}{G_{MN}} \sum_{i=1}^N \frac{m_i}{M} \cdot \frac{\lambda_i}{\sum_{k=1}^N \lambda_k} \right). \quad (4)$$

Proof. For a given state z such that $m_i < M$ and $0 < m_j$, the detailed balance equations of this system for any r becomes

$$\pi(\bar{m})P(\bar{m}, \delta_{ij}\bar{m}) = \pi(\delta_{ij}\bar{m})P(\delta_{ij}\bar{m}, \bar{m}), \quad (5)$$

$$\text{where } P(\bar{m}, \delta_{ij}\bar{m}) = \frac{m_j}{M} \cdot \frac{\lambda_i}{\sum_{k=1}^N \lambda_k} \text{ and } P(\delta_{ij}\bar{m}, \bar{m}) = \frac{m_i+1}{M} \cdot \frac{\lambda_j}{\sum_{k=1}^N \lambda_k}.$$

Here, $\pi(\bar{m})$ is the steady-state distribution of \bar{X} . Thus, (5) becomes

²⁾ Here, we do not assume that the range of the cache size (M) is greater than the number of flows (N). However, for the flows that are not well mixed (that is, burst flows and a large number of flows), ($M < N$), the assumption of independent packet flow identifiers can be weakened; thus, the estimation performance can be degraded. We investigate the estimation performance for these cases in detail by simulations in sections III and V.

$$\pi(\bar{m}) = \pi(\delta_{ij}\bar{m}) \frac{\lambda_j}{m_j} \cdot \frac{m_i + 1}{\lambda_i}. \quad (6)$$

We have found that the detailed balance equation (5) is solved by

$$\pi(\bar{m}) = \prod_{n=1}^N \frac{\lambda_n^{m_n}}{m_n!} / G_{MN}, \quad (7)$$

where the normalizing constant is

$$G_{MN} = \sum_{\bar{m} \in S_{M,N}} \prod_{n=1}^N \frac{\lambda_n^{m_n}}{m_n!}. \quad (8)$$

Thus, we can conclude that the process \bar{X} is time reversible. From (6), (7), and (8) we get

$$\pi(\delta_{ij}\bar{m}) = \prod_{n \neq i,j}^N \frac{\lambda_n^{m_n}}{m_n!} \cdot \frac{\lambda_i^{m_i+1}}{(m_i+1)!} \cdot \frac{\lambda_j^{m_j-1}}{(m_j-1)!} / G_{MN}.$$

Now, let us denote $f(t)$ as the hit frequency, and H as the steady-state hit probability of the cache. Then, H includes the summation of all states as

$$H = \sum_{\bar{m} \in S_{M,N}} \pi(\bar{m}) P_{hit}(\bar{m}, \bar{m}). \quad (9)$$

In practice, H can be estimated by a first-order autoregressive process, defined as

$$f(t) = (1 - \alpha)f(t-1) + \alpha \cdot 1 \{\text{hit at } t\text{-th packet}\}$$

for $0 < \alpha < 1$. In this expression, $1 \{\text{hit at } t\text{-th packet}\}$ represents an indicator function of a cache hit. It is clear that H is a limiting point of the process f , that is, $\lim_{t \rightarrow \infty} f(t) \equiv H$. We assume that the choice of α is such that $f(t)$ converges faster than the rate of change of N . (TCP estimates the round-trip time using an autoregressive process, too.) Here, α plays a role as a time constant that determines the speed of the model to reach the steady-state. Finally, from (2), (7), and (9), we get (4) and thus, prove theorem 1. \square

Observe that if each λ_i in the summand of the numerator of (4) is replaced by the maximum value of λ_{\max} , then H is less than or equal to $\lambda_{\max} / \sum_{k=1}^N \lambda_k$. Similarly, if each λ_i in the summand of the numerator of (4) is replaced by the minimum value of λ_{\min} , then H is greater than or equal to $\lambda_{\min} / \sum_{k=1}^N \lambda_k$. Therefore, we have the following two corollaries.

Corollary 1.

$$\lambda_{\min} / \sum_{k=1}^N \lambda_k \leq H \leq \lambda_{\max} / \sum_{k=1}^N \lambda_k. \quad (10)$$

Corollary 2. If all the arriving rates, λ_i , are equal for all N , then the upper and lower bounds of H in (10) are equal, leading to $H=1/N$.

This expression shows that the number of active flows, N , can be computed from the steady-state hit frequency, H . We calculated the steady-state hit frequency and the number of flows using (4) and compared them with the simulation results. Due to

Table 1. Estimated number of flows.

Number of flows used	Memory size	Estimated number by (4)	Estimated number by simulation		
			r=1.0	r=0.25	r=0.01
10	5	10	31.81	12.59	10.33
	10	10	16.20	11.70	10.21
	40	10	11.52	10.14	9.91
	80	10	10.57	9.42	10.0
50	25	50	161.78	63.31	52.86
	50	50	88.01	56.64	51.35
	200	50	59.65	50.24	51.78
	400	50	52.73	52.13	50.86
100	50	100	334.71	137.33	109.76
	100	100	182.71	115.32	108.10
	400	100	112.58	104.56	106.11
	800	100	116.25	110.46	105.96

the large state space ($(M+N-1)C_M$), we first investigated the accuracy of our model with a relatively small cache memory (10 cache lines) and a small number of flows (10 or less) by enumerating the entire state space. We then extended the validation for large cache size (up to 800 cache lines) and a greater number of flows (up to 100) by using various state-space truncation techniques.

When the same arrival rate was used for all flows, the estimation using (4) was exactly the same as the actual number of flows. Table 1 shows a comparison of the results of the estimation by (4) and the simulation results. (We used the NS-2 simulator, and the simulation environment is described in section III.) The results indicate that the model is quite robust in estimating the number of flows. However, the simulation results show that the accuracy of estimation depends on the cache size, M , and the replacement probability, r . Larger M , comparable to the number of flows, and a smaller r help improve estimation accuracy.

III. Limitations of SRED

To examine the capability of SRED in estimating the number of active flows, we performed simulations using a dumbbell topology with a common link bandwidth of 45 Mb/s. In the simulations, each source node initiates data transmissions using FTP over TCP between 0 to 1 second. Although there are many versions of TCPs, such as Reno, Sack, and Vegas [13]-[15], we use TCP Reno [16], which has a basic congestion avoidance mechanism, with the maximum window size of 45 packets since the main focus of the research is traffic measurement/estimation. Each intermediate node has a buffer size of 600 packets, while the packet size is fixed at 1 kb.

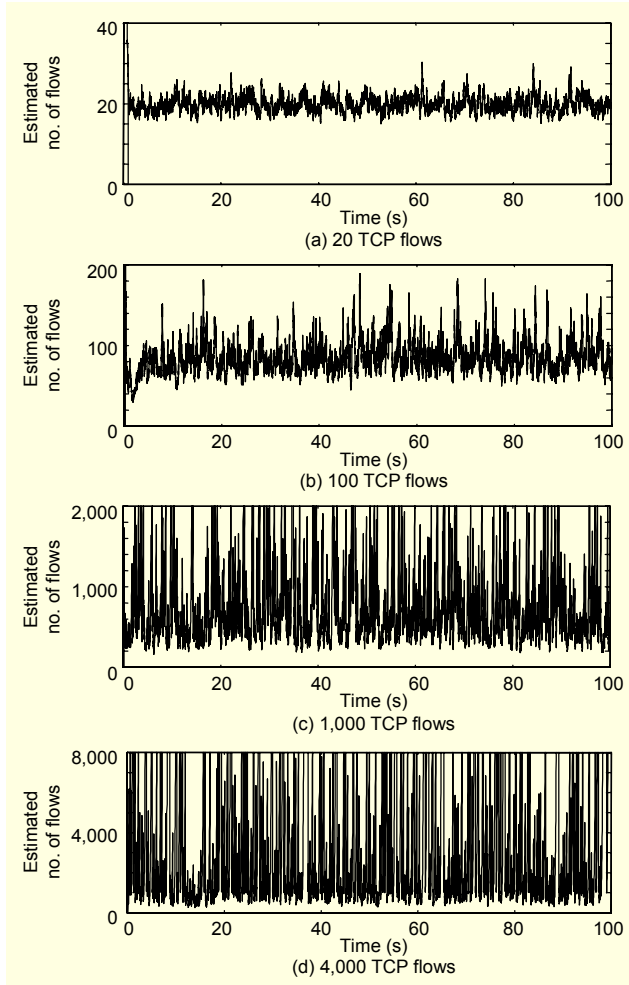


Fig. 2. Estimated number of flows with SRED.

SRED is deployed at the intermediate node to estimate the number of active flows with a cache size of 1,000 lines, $r = 0.25$ and $\alpha = 0.001$, as used in [4].

We investigate two important factors that most affect the estimation performance of SRED: stability of estimation and the impact of non-responsive flows.

1. Stability

Figure 2 shows the estimated number of active flows with SRED when 20, 100, 1,000, and 4,000 flows are used. The estimation capability of SRED is quite accurate with a small number of flows, but it greatly fluctuates as the number of flows increases. The fluctuation is due to the low hit probability when many flows are used.

2. Impact of Non-responsive Flows

The effect of non-responsive flows (generally UDP), when mixed with TCP flows has been extensively studied in the context of active queue management (AQM) schemes [17], [18]. These

Table 2. Impact of non-responsive flows in estimating the number of flows.

Fraction of misbehaving flows in total workload	Estimated number of flows	
	$\lambda_m = 2 \cdot \lambda_c$	$\lambda_m = 3 \cdot \lambda_c$
0 %	10.000	10.000
10 %	9.309	8.001
20 %	9.000	7.541
30 %	8.897	7.529
40 %	8.907	7.712
50 %	9.902	8.000

studies tried to detect the non-responsive flows with a minimum amount of per-flow information. Another approach, called stochastic fair blue (SFB) [19], controls non-responsive flows without using per-flow information via a group of hash tables.

A problem of hash-based techniques such as SFB and SRED is that when many non-responsive flows are present in a network, the hash table is contaminated by these flows, and performance is significantly degraded. We demonstrate the impact of non-responsive flows using the SRED mathematical model in Table 2 and a simulation study presented in section V.

As shown in Table 1, we used a cache size of 10 lines with 10 flows and set the arrival rate of non-responsive flows (λ_m) to 2 and 3 times that of the conforming flows (λ_c) to mimic the behavior of non-responsive flows in (4).

The small memory size and number of flows are not enough to capture the exact effect of non-responsive flows, but it helps us to predict the tendency when many flows are used. Table 2 shows that as the proportion of non-responsive flows increases, the number of active flows is underestimated. When non-responsive flows become a dominant part of the traffic, the estimated number starts to recover the underestimation that results from memory contention among non-responsive flows. These results show a trend similar to that of the simulation result for 500 flows (TCP + UDP) presented in section V (Fig. 6).

In summary, SRED exhibits unstable estimation with a large number of flows and underestimation in the presence of non-responsive flows.

IV. The Proposed Hash-Based Two-Level Caching Scheme (HaTCh)

Based on the discussions in the previous section, we propose the HaTCh flow estimation scheme to minimize SRED's innate problems. The HaTCh scheme consists of two parts: a hashing scheme to stabilize estimation, and a two-level caching scheme to isolate the non-responsive flows that contaminate the zombie

list and lead to underestimation.

1. Hash-Based Estimation

The key idea of the hashing scheme is based on the observation that SRED's hit probability is low when there are many flows. This problem can be alleviated by using a hashing scheme. To implement hashing, the single cache memory (zombie list) is partitioned into k small chunks, called subcaches. Whenever a packet arrives, the packet is hashed into a subcache using the source and destination addresses, and a cache line is randomly selected for comparison from the subcache. A connection is always hashed to the same subcache with this technique. Each subcache maintains its own hit frequency and estimated number of flows. The estimated number of flows per subcache is aggregated to find the total number of estimated flows. The performance of the hash-based estimation may degrade when most active flows are hashed into one or two subcaches, but this can be alleviated by periodically scattering the hash function as noted in [20]. When all the flows have the same sending rate and round-trip time, the hit probability of a flow is $1/N$ under SRED; however, the hit probability increases to k/N when k subcaches are used in HaTCh. This improved hit probability helps to achieve accurate and stable flow estimation.

2. Two-Level Caching Scheme

Non-responsive flows tend to send more packets than conforming flows, and this increases the hit rate. Therefore, developing an efficient scheme to isolate *excess* packets from the memory is the key for accurate estimation. We accomplish this through a two-level cache design.

Figure 3 shows the basic organization of HaTCh, which combines hashing and two-level caching. The structure of the two-level caching proposed here is similar to that of the general two-level caching scheme. The major differences are that the inclusion property is not necessarily satisfied in the two-level cache model, and the L1 and L2 cache update operations are also different. The L1 cache is smaller than the second level L2 cache, and each of the two caches is divided into k subcaches (blocks). Note that there is a corresponding subcache in L2 for each subcache in L1.

An arriving packet is hashed into one of the L1 subcaches using the source and destination addresses. Then, a randomly selected cache line from the L1 subcache is compared with the arriving packet. If there is an L1 cache hit (case 1 in Fig. 3), the hit count of the cache line is increased by 1, but the hit frequency of this subcache remains the same. Otherwise, the corresponding L2 subcache is selected. Then, a randomly selected L2 cache line in the corresponding subcache is compared with the arriving packet. If there is an L2 cache hit (case 2 in Fig. 3), the previously selected L1 cache line is updated with this L2 cache line. The hit

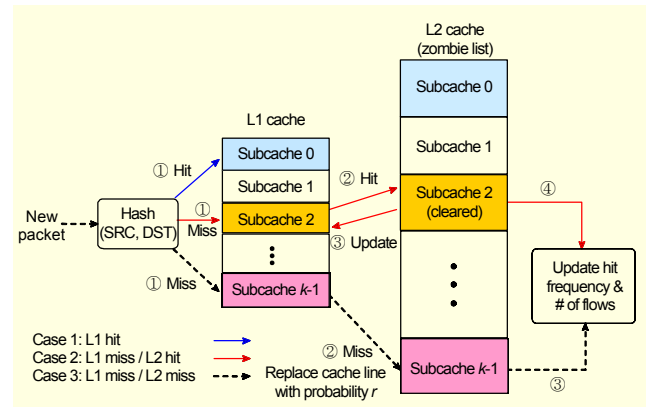


Fig. 3. HaTCh (hash-based two-level caching) architecture.

count of the L1 cache line is set to 1, and the L2 cache line is then cleared. If the L2 cache misses (case 3 in Fig. 3), the L2 cache line is replaced with the arriving packet with probability r .

Whether there is a hit or miss in the L2 cache, the hit frequency and the estimated number of flows for the subcache is recalculated followed by the total number of flows. The sequence of operations for the three cases is shown by the solid, dotted, and dashed lines in Fig. 3.

The key features of HaTCh are following. First, HaTCh takes advantage of the improved hit probability by hashing in both L1 and L2 caches. This stabilizes the estimation process of HaTCh. Second, the hit frequency is recalculated only when there is an L1 cache miss. The L1 cache is updated only when a flow hits the L2 cache; thus, the L1 cache is generally shared by the flows that hit the L2 cache. Filtering these flows to the L1 cache provides a fair chance for all the flows to update the hit frequency. Third, on an L2 cache hit, the selected cache line is cleared to yield room for the subsequent conforming flows using a mechanism called L2 cache cleaning. Ideally, the L2 cache (zombie list) should be shared uniformly among all competing flows to yield an accurate flow estimation. Although the non-responsive flows are filtered in the L1 cache, the flows that missed the L1 cache fill the L2 cache more aggressively than conforming flows. Therefore, the cleaning mechanism in the L2 cache also contributes to fair sharing of the L2 cache.

3. Preliminary Model of the HaTCh Scheme

In this section, we present a mathematical model for the proposed HaTCh scheme to demonstrate its effectiveness in isolating non-responsive flows. Unlike the SRED model previously presented, where we were able to compute the hit frequency from the steady-state distribution of the cache lines occupied by each flow, here, we simply show how the proposed scheme controls the non-responsive flows. The model for HaTCh is extremely complex; thus, state-space enumeration and

computation is expensive. Instead of focusing on the solution of the steady-state probabilities, we show through a Markov model why two-level caching is more effective than SRED.

The model extends the single cache design to capture the two-level cache memory with M_1 lines for the L1 cache and M_2 lines for the L2 cache ($M_1 \ll M_2$), and N independent flows. If we assume that (X_i, Y_i) are the numbers of cache lines with the flow ID i in the L1 and L2 caches respectively, and (\bar{X}, \bar{Y}) is a pair of vectors representing the number of cache lines in L1 and L2 occupied by each flow, then (\bar{X}, \bar{Y}) represents the discrete state model of the system. Now the state space for \bar{X}, \bar{Y} can be defined as

$$\begin{aligned}\bar{X} &\in \left\{ \bar{c} = (c_1 \cdots c_i \cdots c_N)^T \mid 0 \leq c_i \leq M_1, \sum_{i=1}^N c_i = M_1 \right\} \equiv S_X, \\ \bar{Y} &\in \left\{ \bar{m} = (m_1 \cdots m_i \cdots m_N)^T \mid 0 \leq m_i \leq M_2, \sum_{i=1}^N m_i \leq M_2 \right\} \equiv S_Y, \\ \text{and } S_{M_1, M_2, N} &\equiv S_X \times S_Y.\end{aligned}$$

Let us assume an arriving packet has a flow ID i , a randomly selected cache line from the L1 cache has a flow ID j , and a randomly selected cache line from the L2 cache has a flow ID k . For a given cache state (\bar{c}, \bar{m}) , if there is an L1 cache hit or miss in both the caches and the cache line is not replaced in the L2 cache, the number of cache lines for each flow in both caches remains the same, (\bar{c}, \bar{m}) . On the other hand, if there is an L1 cache miss and L2 cache hit, the state changes to the next state $\gamma_{ij}(\bar{c}, \bar{m})$, where $c_i = c_i + 1$, $c_j = c_j - 1$, and $m_i = m_i - 1$. If both L1 and L2 caches incur misses, and the L2 cache is replaced with the replacement probability of r , the state changes to another state $\eta_{ij}(\bar{c}, \bar{m})$, where $m_i = m_i + 1$ and $m_k = m_k - 1$.

For a given state (\bar{c}, \bar{m}) , the probability of an L1 cache hit is the product of the probability that an arriving packet has the flow ID i and the probability that a randomly selected L1 cache line has the same flow ID. Accordingly, the probability of an L1 cache miss and L2 cache hit is the product of the probability that an arriving packet has the flow ID i , the probability that a randomly selected L1 cache line has a flow ID other than i , and the probability that a randomly selected L2 cache line has the flow ID i . Similarly, the probability of both L1 and L2 cache misses is computed by considering the probability that both the L1 and L2 cache lines have different flow IDs other than i .

Now, the transition probability of the two-level cache model can be written for an L1 cache hit or for miss in the both caches and without replacement in the L2 cache as

$$\begin{aligned}P((\bar{X}, \bar{Y})(t+1) = (\bar{c}, \bar{m}) \mid (\bar{X}, \bar{Y})(t) = (\bar{c}, \bar{m})) \\ \equiv P_{hit}((\bar{c}, \bar{m}), (\bar{c}, \bar{m})) + P_{miss}((\bar{c}, \bar{m}), (\bar{c}, \bar{m}))(1-r) \\ = \sum_{i=1}^N \frac{c_i}{M_1} \frac{\lambda_i}{\sum_{z=1}^N \lambda_z} + \sum_{\substack{i,j,k=1 \\ i \neq j, i \neq k}}^N \frac{c_j}{M_1} \frac{m_k}{\sum_{z=1}^N m_z} \frac{\lambda_i}{\sum_{z=1}^N \lambda_z} (1-r).\end{aligned}\quad (11)$$

For an L1 miss and an L2 hit, the state transition probability becomes

$$\begin{aligned}P((\bar{X}, \bar{Y})(t+1) = \gamma_{ij}(\bar{c}, \bar{m}) \mid (\bar{X}, \bar{Y})(t) = (\bar{c}, \bar{m})) \\ \equiv P((\bar{c}, \bar{m}), \gamma_{ij}(\bar{c}, \bar{m})) \\ = \sum_{\substack{i,j=1 \\ i \neq j}}^N \frac{c_j}{M_1} \frac{m_i}{\sum_{z=1}^N m_z} \frac{\lambda_i}{\sum_{z=1}^N \lambda_z} \\ = \sum_{i=1}^N (1 - \frac{c_i}{M_1}) \frac{m_i}{\sum_{z=1}^N m_z} \frac{\lambda_i}{\sum_{z=1}^N \lambda_z}.\end{aligned}\quad (12)$$

For misses in both caches and the L2 cache line being replaced with probability r , the equation becomes

$$\begin{aligned}P((\bar{X}, \bar{Y})(t+1) = \eta_{ij}(\bar{c}, \bar{m}) \mid (\bar{X}, \bar{Y})(t) = (\bar{c}, \bar{m})) \\ \equiv P((\bar{c}, \bar{m}), \eta_{ij}(\bar{c}, \bar{m}))r \\ = \sum_{\substack{i,j,k=1 \\ i \neq j, i \neq k}}^N \frac{c_j}{M_1} \frac{m_k}{\sum_{z=1}^N m_z} \frac{\lambda_i}{\sum_{z=1}^N \lambda_z} r.\end{aligned}\quad (13)$$

Note that these equations are derived using the same context that we used for SRED model. Therefore, for the two-level cache under HaTCh, we can prove the following theorem following the idea of (8).

Theorem 2. Under the assumption of independent packet flow identifiers, the hit frequency calculated by a first-order autoregressive process for the two-level cache system (HaTCh) converges to

$$\begin{aligned}H &= \sum_{(\bar{c}, \bar{m}) \in S_{M_1, M_2, N}} \pi(\bar{c}, \bar{m}) P((\bar{c}, \bar{m}), \gamma_{ij}(\bar{c}, \bar{m})) \\ &= \sum_{(\bar{c}, \bar{m}) \in S_{M_1, M_2, N}} \pi(\bar{c}, \bar{m}) \sum_{i=1}^N (1 - \frac{c_i}{M_1}) \frac{m_i}{\sum_{z=1}^N m_z} \frac{\lambda_i}{\sum_{z=1}^N \lambda_z}\end{aligned}\quad (14)$$

in the steady-state.

In this equation $\pi(\bar{c}, \bar{m})$ represents the steady-state distribution of (\bar{X}, \bar{Y}) . Unlike the SRED model, it is difficult to find a closed form expression for $\pi(\bar{c}, \bar{m})$. Therefore, instead of computing the state probability distribution, we use the HaTCh model (as in (14)) to explain how HaTCh isolates non-responsive flows and improves the estimation accuracy as compared to SRED.

The steady-state hit frequency is determined by the steady-state cache line distribution and the steady-state hit probability as shown in (4) and (14). In (4), for the SRED model, the effect of non-responsive flows is magnified by both the arrival rate of the non-responsive flows, λ_b , and the number of cache lines occupied by the flows, m_i/M , which is proportional to the arrival rate. This leads to underestimation by SRED when non-responsive flows are present.

In contrast, (14) clearly indicates how HaTCh effectively

isolates non-responsive flows and yields more accurate estimation. First, HaTCh clears the selected L2 cache line on an L2 cache hit to create more room for the subsequent conforming flows, and this contributes to a fair distribution of the L2 cache lines for all active flows. The term $m_i / \sum_{z=1}^N m_z$ in (14) captures this because it represents the probability that the flow ID in the L2 cache is i . Second, the distribution of the L1 cache lines, which is occupied by other flows, the $(1-c_i/M_1)$ term in (14) mitigates the effect of arrival rate of the non-responsive flows, λ_i . We validate our claim about the effectiveness of HaTCh through simulation results in the next section.

V. Simulation Results

The HaTCh scheme was simulated to analyze estimation stability and the impact of non-responsive flows. We configured HaTCh with a hash size (k) of 10, and L1 and L2 cache sizes as 100 and 1,000 lines, respectively. The L1 cache size is fixed at 10% of the L2 cache size, which in turn is kept closer to N . The results are summarized below.

1. Stability

Figure 4 shows the estimated number of active flows with HaTCh for two different r values of 0.25 and 0.01 when 20, 100, 1,000, and 4,000 TCP flows are used. The estimated number of flows is remarkably stable compared to Fig. 2 at the cost of a little bit of more delay. However, hatch also shows a tendency to underestimate the number of flows when r is 0.25 as the workloads increase. The accuracy of the estimation is significantly improved when r is 0.01 as was shown in the previous section. Up to 1,000 flows, the estimated number of flows oscillates within a 20% range with HaTCh. As the number of flows increases, the fluctuation increases. Also, the number of flows is underestimated due to traffic bursts and insufficient cache size.

Theoretically, SRED-like mechanisms (including HaTCh) can keep track of N/r flows as discussed in [4]. We observed that if N is greater than the number of L2 cache lines, M_2 , the HaTCh performance starts to degrade, that is, oscillation exceeds 20% ranges. However, unlike SRED, HaTCh shows smooth (damped) oscillation even for 8,000 TCP flows, but the oscillation gradually increases as the number of flows increases from 1,000 to 8,000. Since the memory requirement for each cache line is very marginal (about 16 B), we believe that the L2 cache can accommodate a large number of flows (16 MB L2 cache can support 1 million flows within 20 % error bound).

Assuming a perfect hash function, the optimal hash size could be the same as the L2 cache size, since all the flows could be hashed into exactly one cache line. In the following

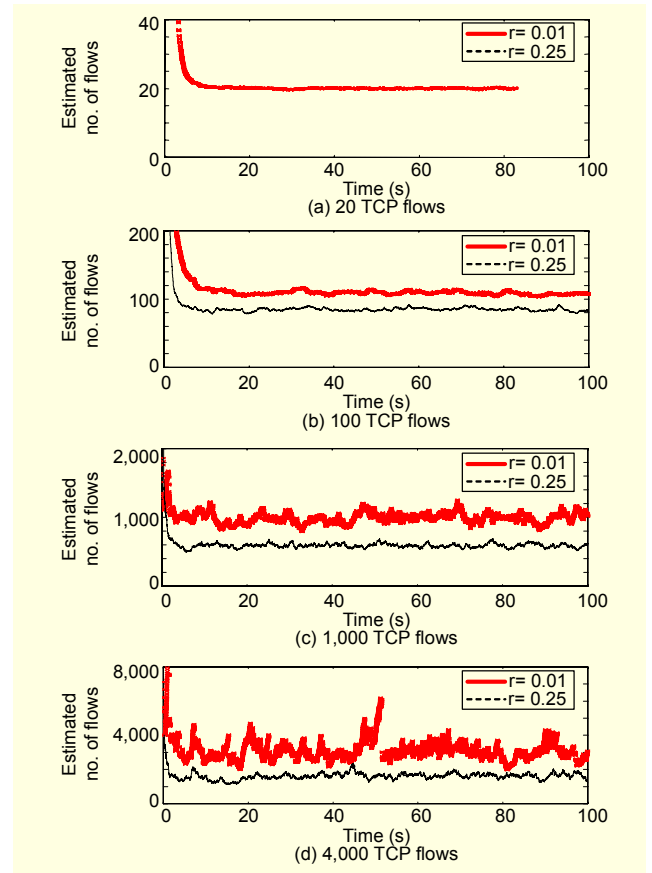


Fig. 4. Estimated number of flows with HaTCh.

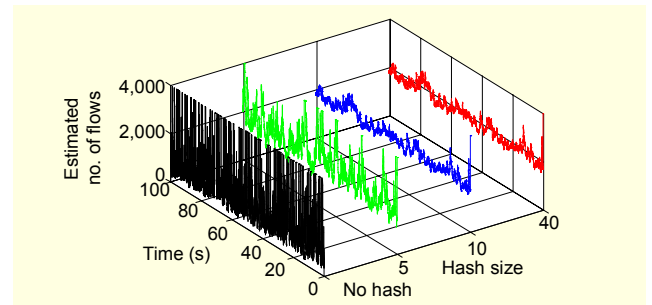


Fig. 5. Impact of hash size on the estimated number of flows ($r=0.01$).

simulation, we examined the impact of k on the estimation behavior of HaTCh with 2,000 TCP flows, while the total memory requirement remains the same (1,000 cache lines). Figure 5 depicts the estimation results of HaTCh with different hash sizes. As noted in the comparison between SRED and HaTCh, hashing with the L2 cache cleaning mechanism generally degrades the response time, especially when a small number of flows is used. The delay is due to the time required to fill the L2 cache lines. Although the hash size of 40 outperforms all other cases in terms of accuracy, it also has the longest response time. We leave this as a design study that optimizes the

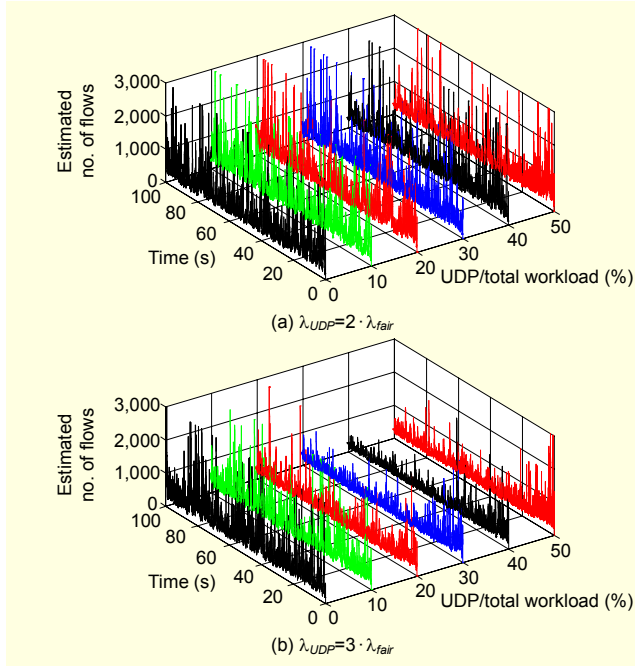


Fig. 6. Impact of non-responsive flows on estimation of the number of flows with SRED.

accuracy and response time of the system, and thus we use $k=10$ for the following simulations.

2. Impact of Non-responsive Flows

To demonstrate the effect of non-responsive flows on HaTCh performance, we carried out simulations with different sending rates of non-responsive UDP flows. We also varied the UDP traffic workload from 0 to 50% of the total workload. Figure 6 shows the results of SRED estimation with 500 flows. In Fig. 6(a), we set the sending rate of the non-responsive flows (λ_{UDP}) to 2 times the fair share of the link bandwidth (λ_{fair}). Although the estimated number of flows goes down slightly, the fluctuation is not alleviated. The impact of non-responsive flows is more pronounced when we increase the UDP sending rate to 3 times the fair share of the link bandwidth shown in Figure 6(b). Here, SRED begins to underestimate the number of flows as the UDP workload increases up to 30%, and the estimation later recovers as the UDP flows becomes a dominant part of the traffic (40 and 50%).

We repeated the same experiments using HaTCh. Note that we used different scaling factors in the graphs to show the results. In Fig. 7(a), HaTCh's estimation is remarkably stable, and the non-responsive flows are successfully isolated as expected. HaTCh shows the same performance up to 20% of the UDP workload shown in Fig. 7(b) as well. As the UDP workload increases, HaTCh also shows gradual performance degradation since a large number of excess packets from non-responsive flows cannot be captured in the L1 cache. However, the estimation error is significantly reduced with HaTCh as compared to SRED.

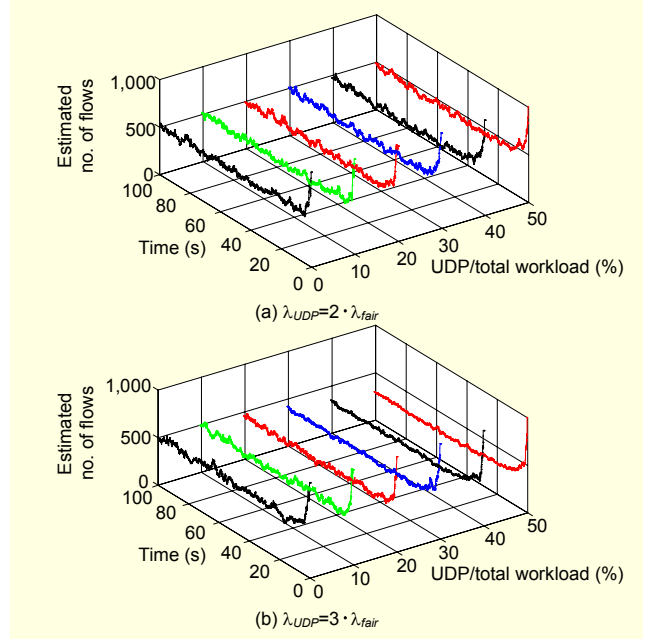


Fig. 7. Impact of non-responsive flows on estimation of the number of flows with HaTCh.

The performance of HaTCh is not significantly affected by the size of the L1 cache as long as the number of L1 cache lines is greater than the number of non-responsive flows. The size of the L1 and L2 cache lines can be determined based on the target number of non-responsive flows and the target number of active flows. In our experiments, we set the L1 cache size to 100 lines to control up to 100 non-responsive flows. This configuration provides the best performance for our simulation environment, where HaTCh effectively isolates up to 100 UDP flows (20% of total workload) as shown in Fig. 7(b).

The impact of TCP sources with heterogeneous round-trip times (RTTs) is an important factor that affects the performance of AQM schemes. TCP flows with short RTTs slow down the transmission rates quickly after congestion notification (packet drop) of an AQM scheme, but they recover quickly compared to those with long RTTs. Therefore, an AQM scheme has to react differently for TCP sources with different RTTs to achieve fair sharing of bandwidth. However, flow estimators such as HaTCh have no control on transmission rates of traffic sources. Therefore, HaTCh performance on TCP sources with heterogeneous RTTs can be predicted by investigating the impact of non-responsive flows. The simulation results with TCP sources that experience different RTTs (varied between 60 ms and 540 ms) exhibited the same trend shown in Figs. 6 and 7.

3. Impact of HaTCh Configuration Parameters

Finally, we examine the impact of two major configuration parameters, the second-level cache size M_2 and the hash size k , on the performance of HaTCh. Initially, we set the L2 cache size, M_2 ,

Table 3. Impact of L2 cache size and hash size (relative error).

L2 cache size (M_2)	Hash size (k)	Number of flows per subcache when $M_2 = N$	Ratio of L2 cache size to number of flows		
			1:1	1:2	1:4
200	2	100	0.11618	0.20050	0.29553
	4	50	0.08427	0.13891	0.26878
	8	25	0.06154	0.07143	0.25603
500	5	100	0.11986	0.18994	0.25861
	10	50	0.05328	0.10267	0.25854
	20	25	0.01650	0.08750	0.20056
1000	10	100	0.08164	0.12490	0.24025
	20	50	0.07293	0.09345	0.22993
	40	25	0.02074	0.07790	0.24310
2000	20	100	0.05876	0.106871	0.25973
	40	50	0.02495	0.090319	0.26733
	80	25	0.01679	0.083062	0.28200

to 200 lines, and increased it up to 2,000 lines. We adjusted k to make the number of active flows per subcache constant (100, 50, and 25 flows) for each L2 cache size when the L2 cache size is equal to N . Then, we increased N to two and four times that of M_2 . In all these experiments, the L1 cache size was set at 10% of the L2 cache size because that provided the optimal configuration.

Here, we summarize the performance of HaTCh by presenting the relative error of the estimated number of flows, defined as the ratio of the standard deviation (σ) to the mean (μ) of the estimated number of flows. In Table 3, the relative error was bounded within 11% regardless of the hash size when the L2 cache size and the number of flows were the same. Notably, HaTCh performs better as the number of flows (accordingly the L2 cache size is larger) increases, because a large number of flows multiplexed at the router reduces the impact of burst traffic. A large k helps in the accurate estimation of N at the cost of little additional delay. As the numbers indicate, the relative error is about 6% when k is 20, and is reduced to less than 2% for $k = 80$ when $N = 2,000$.

The trend also indicates that HaTCh can provide more accurate estimate of N if the number of flows is higher than 2,000, which is likely to be true in many practical cases. As N increases to two to four times M_2 , the accuracy of estimation suffers, although the overall trend in the impact of N and k remains valid. For the 1:2 ratio, the error is limited to 8% when $N = 2,000$ and $k = 80$. However, the relative error is high for the 1:4 ratio, which suggests that number of flows should not exceed twice the size of the L2 cache to limit the flow estimation error to 10%. Moreover, the impact of the hash size on the estimation accuracy was reduced because of the presence of more flows contending for the cache lines.

VI. Concluding Remarks

Measuring and monitoring network traffic has been an active area of research in the Internet community for better network management, planning and accurate billing. However, huge traffic volumes and the complexity of memory management required for per-flow monitoring has eluded researchers attempting to find an efficient solution. In this paper, we focused on a flow estimation technique called SRED since it accurately estimates the number of active flows without maintenance of per-flow states.

First, we developed a mathematical framework for analyzing the estimation behavior of SRED. The model showed that the steady-state hit frequency of the SRED cache model can be used to estimate the number of active flows. The model is accurate enough to capture the effect of non-responsive flows in the estimation. We, then, proposed a modification of SRED, called HaTCh, that uses hashing and a two-level caching mechanism to alleviate the drawbacks of SRED. A preliminary instance of the proposed scheme was presented to demonstrate its effectiveness. Also, we conducted extensive simulations for an in-depth performance analysis of both the schemes. The proposed HaTCh scheme improves estimation accuracy and stability compared to that achieved by SRED. It also improves the robustness of the estimation by effectively isolating the non-responsive flows to avoid cache contamination.

The proposed scheme is practically viable since the two caches can be implemented using standard hardware. A standard configuration of HaTCh could be the following: (i) L2 cache size = the target number of flows supported by the link (N); (ii) L1 cache size = 10% of L2 cache size or the target number of non-responsive flows; (iii) $r = 0.01$, and (iv) k is set as large as possible based on the implementation complexity.

Our future work will involve development of a complete AQM mechanism and security measure based on the HaTCh concept. Also, we plan to develop a self-tuning strategy for HaTCh, which will include consideration of other system and design parameters.

References

- [1] Architecture for IP Flow Information Export, Internet Draft, Available from <http://www.ietf.org/internet-drafts/draft-ietf-ipfix-architecture>.
- [2] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *Proc. ACM SIGCOMM*, Aug. 2002, pp. 323-336.
- [3] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, "Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement," *Proc. IEEE INFOCOM*, Mar. 2004, pp. 1762-1773.
- [4] T. Ott, T. Lakshman, and L. Wong, "SRED: Stabilized RED," *Proc. IEEE INFOCOM*, Mar. 1999, pp. 1346-1355.
- [5] S. Yi, X. Deng, G. Kesidis, and C.R. Das, "A Dynamic Quarantine Scheme for Controlling Unresponsive TCP Flows," Pennsylvania

State University Technical Report CSE04-004, Feb. 2004.

- [6] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated Worm Fingerprinting," *Proc. the 6th ACM/USENIX Symp. Operating System Design and Implementation (OSDI)*, Dec. 2004, pp. 45-60
- [7] C. Estan and G. Varghese, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," *Proc. ACM Internet Measurement Conf.*, Oct. 2003, pp. 153-166.
- [8] V. Paxson, "End-to-End Routing Behavior in the Internet," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, Oct. 1997, pp. 601-615.
- [9] Network simulator (NS), on-line document, available from <http://www.isi.edu/nsnam>.
- [10] M. Chan and M. Hamdi, "An Active Queue Management Scheme Based on a Capture-Recapture Model," *IEEE Journal on Selected Areas in Comm.*, vol. 21, no. 5, May 2003, pp. 572-583.
- [11] A. Kuzmanovic and E.W. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks: The Shrew vs. the Mice and Elephants," *Proc. ACM SIGCOMM*, Aug. 2003, pp. 75-86.
- [12] A. Tang, J. Wang, and S.H. Low, "Understanding Choke," *proc. INFOCOM*, San Francisco, Apr. 2003, pp. 114-124.
- [13] W.R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison Wesley.
- [14] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Option," Internet draft, work in progress, 1996.
- [15] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *Proc. ACM SIGCOMM*, Oct. 1994, pp. 24-35.
- [16] V. Jacobson, "Congestion Avoidance and Control," *Proc. ACM SIGCOMM*, Aug. 1988, pp. 314-329.
- [17] D. Lin and R. Morris, "Dynamics of Random Early Detection," *Proc. ACM SIGCOMM*, Sept. 1997, pp. 127-137.
- [18] R. Mahajan and S. Floyd, "RED-PD: Controlling High Bandwidth Flows at the Congested Router," *the 9th Int'l Conf. Network Protocols*, Nov. 2001, pp. 192-201.
- [19] W. Feng, D.D. Kandlur, S. Debanjan, and K. Shin, "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness," *Proc. IEEE INFOCOM*, Apr. 2002, pp. 1520-1529.
- [20] P. McKenney, "Stochastic Fairness Queueing," *Proc. IEEE INFOCOM*, Mar. 1990, pp. 733-740.



Sungwon Yi received the MS and PhD degrees in computer science and engineering, from Pennsylvania State University in 2004 and 2005, respectively. Before studying at Pennsylvania State University, he worked for LG-CNS as a system engineer. Since 2005, he has been with ETRI, Korea, where he is a researcher in the Information Security Research Division. His research interests include the areas of computer networks, with emphasis on Internet congestion control and QoS, network security, and mobile computing.



Xidong Deng received her BS degree from the University of Science and Technology of China, Hefei, China, in 1999; and her PhD degree in computer science and engineering from Pennsylvania State University in 2004. She joined St. Cloud State University, Minnesota, in 2004 as an assistant professor with a joint appointment with the Department of Computer Science and the Department of Electrical and Computer Engineering. Her teaching and research interests are computer networks, distributed systems, and performance evaluation. Her recent work has focused on Internet QoS, resource management and congestion control, and network security. She is a member of IEEE and ACM.



George Kesidis received his MS and PhD degrees in EECS from the University of California, Berkeley, in 1990 and 1992, respectively. He was a professor in the E&CE department of the University of Waterloo, Canada, from 1992 to 2000. Since April 2000, he has taught in both the CS&E and EE departments of Pennsylvania State University. His research experience spans several areas of computer/communication networking, including security, incentive engineering, efficient simulation, and traffic engineering. He served as TPC co-chair of IEEE INFOCOM 2007 and is currently an associated editor for ACM TOMACS and IEEE Communications Surveys and Tutorials. He is a senior member of the IEEE.



Chita R. Das received the MSc degree in electrical engineering from the Regional Engineering College, Rourkela, India, in 1981; and the PhD degree in computer science from the Center for Advanced Computer Studies, University of Louisiana, Lafayette, in 1986. Since 1986, he has been with Pennsylvania State University, where he is currently a professor with the Department of Computer Science and Engineering. His main areas of interest are parallel and distributed computer architectures, cluster computing, mobile computing, Internet QoS, multimedia systems, performance evaluation, and fault-tolerant computing. He has served on the editorial boards of the IEEE Transactions on Computers and IEEE Transactions on Parallel and Distributed Systems. Dr. Das is a Fellow of the IEEE and a member of the ACM.