Data stream algorithms for scalable bandwidth management

Sreenivas Gollapudi
Oracle Corporation / SUNY at Buffalo
Email: sreenivas.gollapudi@oracle.com

D. Sivakumar
IBM Almaden Research Center
Email: siva@almaden.ibm.com

***Abstract*— We propose an efficient and scalable scheme for bandwidth reservation and monitoring. Our scheme is based on a reserve-and-refresh strategy [14], [9], where each flow is periodically refreshes its initial reservation.**

We propose novel algorithms to handle various forms of misbehavior, e.g., attempting to refresh more than what was reserved (control plane), exceeding reservations (data plane). Our solutions are based on *data stream algorithms* that are extremely efficient in terms of memory requirements and time required to process each packet. Specifically, we compute very short *sketches* of packet traffic with which we can provably guarantee that no more than a tiny fraction of the bandwidth is lost to misbehaving flows.

Since our solutions are robust, incrementally deployable, and have very low time/space requirements, we believe they are ideally suited for supporting QoS, flow and congestion control, and more generally, for bandwidth management in active network architectures.

I. Introduction

As the variety of Internet applications evolves, supporting Quality of Service (QoS) requirements has become a major challenge. To address this, three broad approaches have emerged: Integrated Services (IntServ [3]), Differentiated Services (DiffServ [2]), and Stateless CORE (SCORE [13], [14]). Roughly speaking, IntServ proposes a solution where all routers carry per-flow information, while the latter two approaches differentiate routers at the core of the network (the core routers) from those at the periphery of the network (the edge routers); edge routers carry more detailed per-flow information, while core routers often simply keep track of some aggregate statistics.

The most serious drawback of IntServ is that of scalability, which is due to the fact that all routers carry per-flow information. While Diffserv and SCORE address this concern, they are encumbered by the problem of misbehaving flows: since not all routers maintain per-flow information, it is possible for malicious flows to grab a significant fraction of the bandwidth. Therefore, these approaches can only make weaker QoS guarantees. To alleviate this concern, routers must deploy policing/monitoring mechanisms that detect and punish misbehaving flows.

In this paper, we introduce a novel approach to the problem of misbehaving flows. We study this problem from the viewpoint of *data stream algorithms* (see [1], [10] for recent and comprehensive surveys) and present time- and space-efficient solutions for the problems described above.

The main goal of our solutions is to reduce the overhead and the complexity of the control plane operations; this is achieved by relaxing the requirement that *every* misbehavior is identified. Instead, we wish to ensure that no more than ϵ fraction of the bandwidth is misused by misbehaving flows (for some small $\epsilon > 0$). This viewpoint leads to solutions that are efficient and realistic (in terms of implementation) without compromising much on the quality. Furthermore, when the fraction of flows that misbehave is small, our schemes carry very little overhead—unlike recent proposals based on cryptographic hashing, which require considerable work regardless of whether or not flows are misbehaving.

We rely on the soft-state, reserve-and-refresh framework of [14], [9], which we now describe briefly. We discuss the case of a single link in the network, controlled by a router. The intent is that all the computations we will outline will be carried out by the router; therefore, it is vital that the overhead in the computations is minimal, as is the space used by the computations. The router maintains the total bandwidth W that has been allocated on the outgoing link. When a flow requests to be admitted with a transmission rate r, the router checks if $W + r \leq C$, where C is the capacity of the link; if it is, the flow is admitted and the value of W is updated to become $W + r$. Due to memory constraints, the router cannot maintain a table of how much bandwidth is allocated to each flow; this limitation, together with the fact that flows do not report their termination to the router, makes it impossible to accurately maintain the value W. To address this, each flow will be required to send periodic *refresh* messages on the control plane. The router will use the sum of the values in the refresh messages as an estimate of W. This idea is due to Stoica and Zhang [14]. Each allocation will be refreshed once every T_{ref} time steps, for some parameter T_{ref} that the router prescribes. To handle issues like message loss, jitter, etc., the router will require that each allocation be refreshed n_{ref} times in a "router period" T_{router}, where $T_{\mathrm{router}} = n_{\mathrm{ref}} \cdot T_{\mathrm{ref}}$, and suitably compensate by sufficient overestimation.

The idea of periodically refreshing bandwidth reservations leads to new problems. The *faux refresh* problem deals with a misbehaving flow attempting to refresh an allocation that was never made; when a flow attempts to refresh a reservation made a very long time ago, we call it the *stale refresh problem*. The *extra refresh problem* problem refers to a flow sending too

many refresh messages in a router period, thereby "inflating" its allocation (and also invalidating the correct estimate of W, which could prevent other flows from being admitted). Any misbehavior of these types is classified as *control plane misbehavior*. The second consequence of not maintaining a table of all allocations, called *data plane misbehavior*, refers to the possibility that some flows might exceed their reservations (on the data plane).

We conclude the introduction with a brief outline of related work. In section II, we present the algorithm for detecting faux and stale refreshes; in section III, we present the algorithm for detecting extra refreshes and data plane misbehavior. Section IV describes our experimental results, and section V presents some concluding remarks.

Related work. A simple heuristic for detecting flows that exceed their reservation is random sampling [6], [13], [9], which relies on the idea that the more a flow misbehaves, the more likely it is to be picked in a random sample. Another idea that has been considered is that of policing aggregate flows [7], [9]. Machiraju, Seshadri, and Stoica [9] employ the soft-state approach of [14], where they use cryptographic hash functions to verify the integrity of the refresh messages, and propose an aggregate monitoring algorithm based on recursively partitioning the flows into random groups. The overhead of the control plane checks is large (in terms of packet size and also in terms of router computations) because of verifying cryptographic hashes; the recursive monitoring algorithm for detecting extra refreshes incurs a latency in detecting misbehavior. As we shall see, our algorithms do not suffer from either problem.

II. Faux and stale refresh detection

In this section, we present the algorithm for handling the problems of faux refresh and stale refresh. In our proposal, bandwidth is allocated in integral multiples of a fixed rate w; each allocation of rate w is called a "token." A flow that receives $W_\varphi = k \cdot w$ units of bandwidth will be given k tokens of the form (φ, i), where φ will denote the flow ID, and $1 \leq i \leq k$. The flow φ will then need to refresh each of the k allocations periodically.

Our solutions are based on data stream algorithms for approximately estimating set differences. Let $t \geq 0$ denote some router period. Define the set A_t to consist of pairs (φ, i), where φ is a flow that legitimately underwent admission during some time period $< t$, and $i \leq k$, where k is the number of tokens that φ was issued. Define B_t to consist of the pairs (φ, i) that are actually seen in the refresh messages sent during router period t. Note that the main difficulty is that the router will not be able to store the sets A_t and B_t; rather, it will work with "sketches" of these sets, which are still sufficient to determine the following two conditions:

(C1) $|B_t - A_t| > \epsilon |A_t|$: this means that more than ϵ fraction of the bandwidth is being claimed by misbehaving flows.

(C2) $|A_t - B_t| > \epsilon |A_t|$: this means that at least ϵ fraction of the bandwidth allocated to legitimate flows is no longer used.

Furthermore, it is important for our purposes that the sketches used to detect conditions (C1) and (C2) are easy to update from one router period to the next. Our solution is a dynamic (and asymmetric) variant of the method of Broder *et al* [4], who gave a solution to the problem of estimating symmetric difference of sets.

Consider the abstract problem of estimating set differences. Fix $0 < \epsilon < 1/2$. Let U denote a universe of size N. Let δ denote a small constant, say 0.1 (though this could be easily made much smaller at very little extra cost).

Definition 1 ([4]): A family Π of permutations of a set U is said to be *δ-approximate min-wise independent* if, for every $X \subseteq U$ and every $x \in X$,

$$\left| \Pr_{\pi \in \Pi}[\min\{\pi(y) \mid y \in X\} = \pi(x)] \; - \; \frac{1}{|X|} \right| \; \leq \; \frac{\delta}{|X|}.$$

There are well-known efficient constructions of approximately min-wise independent permutations, where given $u \in U$, computing $\pi(u)$ can be done very efficiently (in time poly-logarithmic in n).

The next lemma is the main analytical argument of our algorithm; its proof is given in the Appendix.

Lemma 1: Let $A, B \subseteq U$, and for a permutation π of U, let $a_{\min}(\pi) = \min\{\pi(a) \mid a \in A\}$ and $b_{\min}(\pi) = \min\{\pi(b) \mid b \in B\}$[1]. Let Π denote a family of δ-approximately min-wise independent permutations. Then

(1) If $|B - A| > \epsilon |A|$, then $\Pr_{\pi \in \Pi}[b_{\min} < a_{\min}] \geq (1 - \delta) \min\{(1/3), (2/3)\epsilon\}$.

(2) If $|A - B| > \epsilon |A|$ and $|B - A| \leq \epsilon |A|$, then $\Pr_{\pi \in \Pi}[a_{\min} < b_{\min}] \geq (1 - \delta)\epsilon(1 - \epsilon)$.

Algorithm. Let $0 < c < 1$ be a confidence parameter, let $0 < \epsilon < 1$ be a loss parameter, and let $T > 0$ be an integer. Let $\ell = \frac{3}{2\epsilon} \ln \frac{T}{c}$. Let $\pi_1, \ldots, \pi_\ell$ be permutations chosen from an approximately min-wise independent family. For $1 \leq j \leq \ell$, and for $t = 0$, define $a_0(j) = \min\{\pi_j(a) \mid a \in A_0\}$. The sketch a_0 is computed from the set A_0 of admitted flows (tokens). For $0 \leq t \leq T$, the algorithm proceeds as follows. Define $b_{t(j)}$ to be $\min\{\pi_{j(b)} \mid b \in B_t\}$. If $b_{t(j)} < a_{t(j)}$ for any j, then declare that more than ϵ fraction of bandwidth is used by misbehaving flows, and invoke a correction mechanism (e.g., punish misbehaving flows, require re-authentication, etc.). Note that this corresponds to detecting condition (C1). If $b_{t(j)} \geq a_{t(j)}$ for every j, *and* $b_{t(j)} > a_{t(j)}$ for at least one j, then let $a_{t+1}(j) = b_t(j)$. Note that this corresponds to the logical condition "(C1) is false and (C2) is true." When this happens, we also request the flow with sketch $a_{t+1}(j)$ to re-authenticate itself.

Correctness. For any $t \geq 0$, assuming we have the correct sketches for A_t and B_t, by Lemma 1, conditions (C1) and (C2) can be detected with probability at least $2\epsilon/3$ with a single permutation π. With ℓ repetitions, any single occurrence of (C1) or (C2) is detected with probability at least $1 - (1 - 2\epsilon/3)^\ell = 1 - (1 - 2\epsilon/3)^{(3/2\epsilon)\ln(T/c)} \geq 1 - e^{-\ln(T/c)} = 1 - c/T$. Specifically, whenever a fraction ϵ of the tokens present in the refresh messages corresponds to faux or stale flows, with

[1]Where π is clear from context, we will simply write $a_{\min}$ and $b_{\min}$.

probability at least $1 - c/T$, the token from a misbehaving flow will become the minimum under at least one of the ℓ permutations chosen. Summing the error probability over T router periods, we have that the total error probability is at most c.

Note also that if the algorithm does not detect condition (C1), it is still possible that $|B_t - A_t|$ is slightly below $\epsilon|A_t|$, say $\epsilon|A_t|/2$. When this happens, the misbehaving flows become "legitimized" since we defined A_{t+1} to be B_t whenever $|B_t - A_t| \leq \epsilon|A_t|$. If this happens repeatedly for many router periods, the cumulative bandwidth lost to misbehaving flows becomes significant. Thus it is crucial that the "watchdog" flows, that is, flows whose sketches are the minimum, be always maintained correctly; this is precisely why we require the flow φ to re-authenticate itself whenever $a_{\min}$ is updated to be $\pi_j(\varphi, i)$ for some j. Over T router periods, the number of such re-authentications required is roughly $|\cup_{t \leq T} A_t|/(\epsilon \widetilde{A})$, where $\widetilde{A}$ is the average size of A_t over the T router periods. In the worst case (where the A_t's are completely disjoint), this is roughly T/ϵ, or $1/\epsilon$ authentications per router period, rather than one authentication per control plane packet. In more realistic cases, we expect this number to be much less.

We summarize this discussion in the following theorem.

Theorem 1: Let $0 < \epsilon < 1$, let $0 < c < 1$, and let $T > 0$ be an integer. Our algorithm for detecting faux and stale refreshes uses space $O(\frac{1}{\epsilon} \log \frac{T}{c} \text{polylog}(N))$, and guarantees that with probability at least $1 - c$, no more than ϵ fraction of the bandwidth is lost during T router periods due to faux refreshes and stale refreshes. Here N denotes the size of the universe of tokens.

III. Algorithm for the data plane and extra refreshes

Our abstract formulation for the data plane problem (and for the control plane problem of extra refreshes) is as follows. Let S_0, S_1 and S_2 denote streams of pairs of the form (ID, VAL), where each ID can appear many times. For a pair $p = (I, V)$, let $\text{ID}(p)$ denote I and $\text{VAL}(p)$ denote V. For an item I, define $\text{TVAL}(I)$ to be $\sum_{p \in S_0 : \text{ID}(p) = I} \text{VAL}(p)$. Call an item I *offending* if $\sum_{p \in S_1 : \text{ID}(p) = I} \text{VAL}(p) > \text{TVAL}(I)$. We will devise an algorithm that first processes stream S_0 and computes a small sketch; then it processes S_1 and updates the sketch further; finally, it processes the stream S_2 and identifies all offending items I that appear in S_2, assuming that the total number of offending items is at most d, where d is a parameter.

We now describe how the abstract formulation applies to the problem of detecting data plane misbehavior. Recall that the set B_t denotes the set of flows that sent refresh messages during router period t, and further assume that the control plane checks have validated that B_t is a legitimate set of flows. The stream S_0 will correspond to the set B_t of flows where each flow φ will yield a pair (I, V), with $I = \varphi$ and $V = W_\varphi$ (its allocation). The stream S_1 consists of pairs (φ, V_t) where φ is a flow that sent data packets in router time period t, and V_t is the total amount of bandwidth used by φ in time period t. Similarly, S_2 is defined with respect to time period $t+1$. Thus, during router time period $t + 1$, our algorithm will identify flows that misbehave during router period t. Here we assume that a flow that misbehaves during period t appears again during time $t + 1$. To identify flows that misbehave in bursts and are otherwise dormant, we can extend the "monitoring" period to R time periods following t, for some parameter R.

Our sketches will be obtained by randomly partitioning the item set into several groups, and tracking the aggregate of the VAL's in each group. The partitioning will be done using a number of random hash functions.

Definition 2 ([5], [15]): Let $\mathcal{D}$ and $\mathcal{R}$ be two sets. A collection $\mathcal{H}$ of functions $\{h : \mathcal{D} \longrightarrow \mathcal{R}\}$ is said to be a *pairwise independent family* if: (1)For every $x \in \mathcal{D}$ and $y \in \mathcal{R}$, $\Pr_{h \in \mathcal{H}}[h(x) = y] = \frac{1}{|\mathcal{R}|}$.

(2) For every $x_1 \neq x_2 \in \mathcal{D}$ and $y_1, y_2 \in \mathcal{R}$, $\Pr_{h \in \mathcal{H}}[h(x_1) = y_1 \text{ and } h(x_2) = y_2] = \frac{1}{|\mathcal{R}|^2}$.

Pairwise independent families $\mathcal{H}$ of hash functions can be constructed so that each function $h \in \mathcal{H}$ can be described by $\log |\mathcal{D}| + \log |\mathcal{R}|$ bits.

Let $\mathcal{I}$ denote the set of possible items, let $n = |\mathcal{I}|$. Let $\delta > 0$ denote a confidence parameter, and let $k = \lceil 2 \log n + \log \frac{1}{\delta} \rceil$. Let $h_1, h_2, \ldots, h_k$ denote k hash functions picked uniformly and independently from a pairwise independent family $\mathcal{H}$ of hash functions from $\mathcal{I}$ into the set $[D] = \{1, \ldots, D\}$, where $D = 2d$. Define, for $j = 1, \ldots, k$ and $b \in [D]$, the sets $\mathcal{I}_j^b = \{I \in \mathcal{I} \mid h_j(I) = b\}$. For each $I \in \mathcal{I}$, define the *signature* of I to be the k-element sequence σ_I defined by $\sigma_I(j) = h_j(I)$. (Note that the j-th element of σ_I tells us which set $\mathcal{I}_j^b$ the item I belongs to.)

The proof of the next lemma is given in the Appendix.

Lemma 2: With probability at least $1 - \delta$, no two items I and I' have the same signature.

The sketch of a stream S_1 of pairs of the form (ID, VAL) will consist of Dk values, $\langle \alpha(j, b) \mid j \in \{1, \ldots, k\}, b \in [D] \rangle$, defined by

$$\alpha(j, b) = \sum_{p \in S_1 : \text{ID}(p) \in \mathcal{I}_j^b} \text{VAL}(p).$$

When we process stream S_2, we compute the quantities $\beta(j, b) = \sum_{p \in S_2 : \text{ID}(p) \in \mathcal{I}_j^b} \text{VAL}(p)$. It is clear that whenever $\beta(j, b) > \alpha(j, b)$, some item $I \in \mathcal{I}_j^b$ has a larger value in S_2 than in S_1. Call a bin (j, b) *corrupt* if $\beta(j, b) > \alpha(j, b)$, and record the list of corrupt bins while processing S_2. Finally, when processing S, for each item $I \in S$, we compute the signature of I with respect to the hash functions $h_1, \ldots, h_k$. If more than $2/3$ of the bins of I are corrupt, we declare I to be an offending item.

If I is an offending item, it will also be true that $\beta(j, b) > \alpha(j, b)$ for *every* j, b such that $I \in \mathcal{I}_j^b$. To account for the possibility that many of I's bins are shared with flows that use less than their allocation, we check if at least two-thirds of the bins are corrupt. If we assume that most flows use their allocated bandwidth, we could set this threshold higher.

It remains to be shown that a non-offending item J is extremely unlikely to be declared offending. Let $C = \max_j |\{b \mid$

$\beta(j,b) > \alpha(j,b)\}|$ denote the maximum of the number of corrupt bins with respect to any of the k hash functions. For any j, the probability that $h_j(J)$ is a corrupt bin is at most $C/D \leq d/D = 1/2$; therefore, the expected number of j's for which $(j, h_j(J))$ is a corrupt bin is at most $k/2$. By Chernoff bounds, the probability that more than $3k/4$ of the bins $(j, h_j(J))$ are corrupt is exponentially small in k, that is, at most $\gamma\delta/n^2$, where γ is some absolute constant.

Finally, we point out how the sketches may be updated when moving from router period t to router period $t+1$. To account for legitimate reservations entering or leaving during or before router period t, we will use an idea similar to the algorithm for faux and stale refreshes: if, in a router period, no extra refresh or data plane misbehavior was detected, we will use the actual traffic (refreshes/data) during that router period, together with any new reservations made, as an estimate of the legitimate reservations for the next router period. We can easily ensure that no more than ϵ fraction of the bandwidth is lost because of this carry-forward assumption. We defer the details to the full paper. It is worth noting here that to adopt this idea, we need to set $D = 2d/\epsilon$, and store the reservations that are made during the most recent router period.

Theorem 2: Let $0 < c < 1$ denote a confidence parameter, and let $d > 0$ be an integer that bounds the maximum number of misbehaving flows. Let $\epsilon > 0$ denote the fraction of bandwidth that we wish to protect. Let $R > 0$ be an integer. Our algorithm uses space $O(\frac{R}{\epsilon}\log\frac{n}{c}(d + \log n + \log d))$, and ensures that with probability at least $1 - c$, the following holds. When the fraction of bandwidth lost due to data plane misbehavior or extra refreshes exceeds ϵ, the misbehaving flows that re-appear within R router time periods following the misbehavior are identified.

IV. Simulations

In this section, we discuss our experimental results via simulations, based on an implementation of our algorithms using the *ns-2* simulation package [11]. Our goal is to demonstrate the feasibility of our algorithms to detect misbehaving flows, and guarantee the QoS for well-behaved flows.

The queues on the links deploy our bandwidth management algorithms. Our simulations were carried out for $T = 1000$ simulated seconds. We consider 1024 flows carrying constant-bit rate traffic (CBR) with a mean packet size of 512 bytes. The inter-packet interval obeys a power-law distribution, where the probability of the inter-packet interval, $1/x$, $x \in [1, 50]$, is proportional to $1/x^2$. This is in accordance with the discovery of [8], [12] that several parameters of interest, especially the packet rate, in Internet traffic measurement obey a power law. Finally, the starting time of the flows is normally distributed with mean $T/2$, and have duration normally distributed around the mean value of $T/4$ seconds.

In our experiments, we do not use min-wise independent permutation constructions with the best known parameters; instead we adopt simple linear permutations, which are provably good approximations (cf. [4]). A flow with ID φ with rate ρ is hashed as $(a\varphi + b\rho + c) \bmod p$, where a, b, and c are

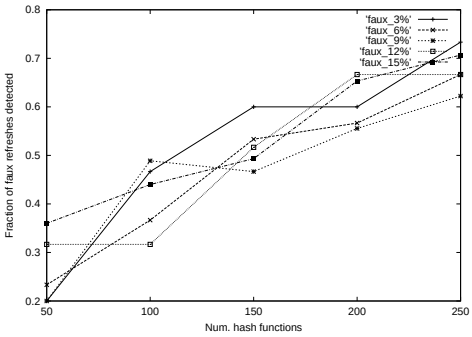

Fig. 1. Detecting Faux Refreshes

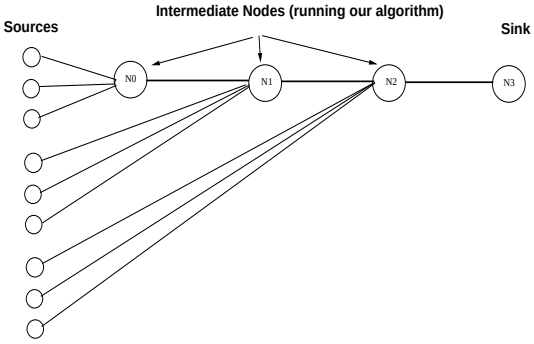

Fig. 2. Network topology for $L = 3$

random entries chosen from the finite field $\mathbf{Z}_p$. We use the same linear hashing scheme (which happens to yield pairwise independence needed for Lemma 2) for the data plane and extra refresh detection algorithms. We fix p to be a large prime number that can be comfortably handled on 32-bit machines.

The first set of experiments is aimed at evaluating the performance of our algorithm for detecting faux refreshes and stale refreshes. Here we study the efficacy of the scheme as a function of the number of hash functions used to create the sketches. Figure 1 shows that the fraction of undetected faux refreshes drops almost linearly with the number of hash functions used. Note that while it is possible that a faux refresh might be missed, our algorithm never mis-classifies a legitimate refresh request as a faux refresh.

The second set of experiments studies the performance of the extra refresh (and data plane) algorithm. The classification error is categorized into two components: (1) $C_m = N_d/N_b$, the fraction of the misbehaving flows undetected; and (2) $C_s = N_m/(N_t - N_b)$, the fraction of well-behaved flows erroneously mis-classified. Here, N_d is the number misbehaving flows undetected, N_m is the number of well-behaved flows erroneously mis-classified, N_b is the total number of misbehaving flows, and N_t is the total number of flows. The objective of this set of experiments is to understand the two classification errors as a function of the number H of hash functions, the size D of the range of the hash functions (cf. Section III), and the number L of links on which the detection algorithms are deployed. The results are summarized

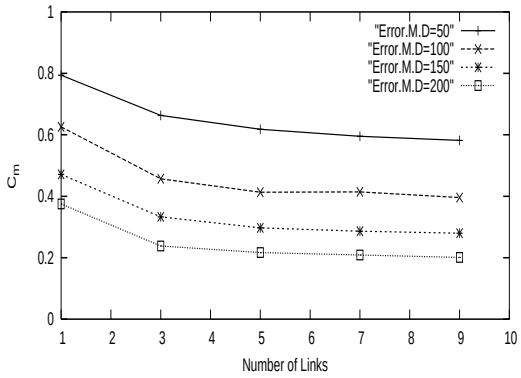

Fig. 3. C_m vs. Number of Routers using the detection algorithms

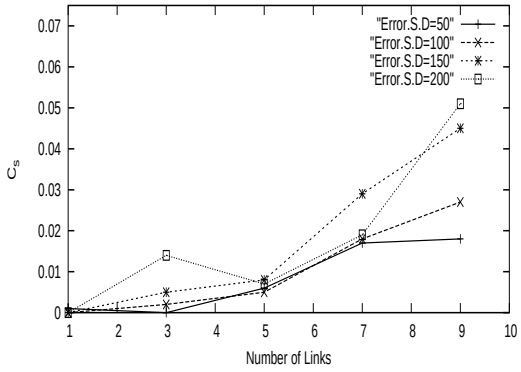

Fig. 4. C_s vs. Number of Routers using the detection algorithms

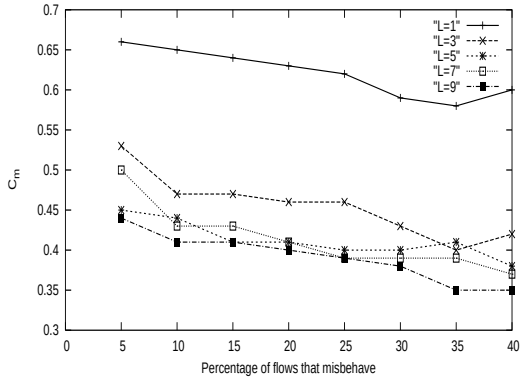

Fig. 5. C_m vs. Percentage of fbws that misbehave

Num. Links	#Hashes/ #Buckets	% Faux ref			% Extra ref	
		N_f	N_e	M_1	N_h	M_2
1	10/50	5.6	13.5	80.9	24.6	75.4
1	10/100	5.1	29.8	65.1	41.3	58.7
1	10/200	6.0	56.9	37.1	65.7	34.3
1	20/50	9.0	11.7	79.3	20.6	79.4
1	20/100	9.1	23.4	67.5	37.1	62.9
1	20/200	9.8	48.1	42.2	62.2	37.8
3	10/50	7.7	24.1	68.2	35.9	64.1
3	10/100	9.3	45.2	45.5	57.5	42.5
3	10/200	9.1	80.5	10.4	77.6	22.4
3	20/50	14.8	17.2	68.1	32.4	67.6
3	20/100	14.6	34.9	50.5	51.5	48.5
3	20/200	15.8	63.6	20.6	73.6	26.4

TABLE I

PERFORMANCE OF THE TWO CONTROL PLANE ALGORITHMS

in Figures 3 and 4, and Table 1. The following sets of values were used: $H \in \{10, 20\}, D \in \{50, 100, 150, 200\}$, and $L \in \{1, 3, 5, 7, 9\}$. The topology for the $L = 3$ case is shown in Figure 2. In general, a topology of $L = n$ has roughly $1/n$ of the flows go through paths of lengths $1, 2, \ldots n$, respectively.

In all experiments, the router period was set to be 0.3 seconds, and the refresh interval was 0.1 seconds; thus well-behaved flows sent roughly three refresh messages per router period. The number of extra refreshes was distributed according to the normal distribution, with the mean refresh interval of a misbehaving flow at 0.05 seconds and standard deviation 0.02.

Figures 3 and 4 illustrate the effect of the number of links on the classification errors. Figure 5 presents the classification error for various values of the fraction of all flows that are misbehaving. From these figures, we make the following salient observations:

(1) Both classification errors, C_m and C_s, drop significantly as we increase the the range size D of the hash functions.

(2) C_m decreases as the number L of links that deploy the detection algorithm increases, while C_s increases marginally. The decrease in C_m is rapid initially, and it becomes slower as the number of links increase. One possible explanation for this observation is the trade off between using a large D vs. a large L — with a large enough D, the first router along any given path identifies most of the misbehaving flows on that path. Routers downstream add marginal value in being able to identify the remaining misbehaving flows. The ability of these routers to identify all the remaining misbehaving flows is dependent on the size of D.

These observations suggest that our algorithm can be incrementally deployed in a network by suitably choosing the number of hash functions and their range at each router. This is further substantiated by another set of experiments, whose results we do not include for lack of space. In these experiments, the parameter D for link was chosen proportional to the traffic it supported.

(3) The number of hash functions has no significant impact on the overall classification error. This suggests the use fewer hash functions per router that map into larger ranges, and have the algorithm deployed in more links.

Table 1 presents a comparative viewpoint of the performance of the two control plane algorithms. Let N_f denote the faux refreshes identified by the faux refresh algorithm; N_e, denote the number of faux refreshes identified by the extra refresh detection algorithm; and let M_1 denote the faux refreshes missed by both. Similarly, we classify the flows that sent extra refreshes into those that were identified by the hash algorithm (N_h), and those that were missed (M_2).

Based on the data in Table 1, we make the following

observations about the performance and complexity of the control plane algorithms.

(1) The success of the faux refresh detection improves with the number of hash functions employed (10 vs. 20)—in fact, almost linearly. Extra refresh detection, on the other hand, seems insensitive to the number of hash functions, and depends more on the size D of the hash function range.

(2) Once again, with more links deploying the faux refresh detection algorithm, the success rate of both algorithms increases.

(3) A large fraction of flows sending faux refreshes are detected by the extra refresh detection algorithm. This should not be surprising, since we may treat a faux refresh as an extra refresh with zero reservation.

We also analyzed the sensitivity of the algorithms to the exact mix of faux and extra refreshes present among the misbehaving flows. We note that the algorithms are quite robust to the variations in the mixture of faux and extra refreshes with a standard deviation of 0.02 around the mean error for a given number of links deploying the algorithms.

V. Concluding Remarks

We have presented a novel approach to bandwidth management for QoS. Our approach is based on data stream algorithms that are very efficient in handling massive data sets. This fits in naturally in the context of networks, where large volumes of IP traffic need to be processed efficiently to support QoS requirements of IP applications.

Our guiding philosophy is that it may be not be important to identify every misbehavior; rather, we seek to make guarantees about the fraction of bandwidth lost due to flow misbehavior, and our algorithms are tunable for this parameter. Moreover, our algorithm can be incrementally deployed in the network depending on the QoS requirements. This is an instance of a general phenomenon, namely providing approximately optimal guarantees leads to a feasible solution for a seemingly intractable problem. Experiments suggest that our algorithms are easy to implement, have low processing/memory requirements, and can be deployed easily in network routers. Our work suggests interesting avenues for further research in the use of data stream algorithms in the areas of flow and congestion control, QoS support, and bandwidth management.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of ACM PODS*, pages 1–16, Madison, Wisconsin, June 2002.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services, 1998. Internet RFC 2475.

[3] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview, 1994. Internet RFC 1633.

[4] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[5] J. L. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[6] C. Estan and G. Varghese. New directions in traffic measurement and accounting, 2001.

[7] W. Feng, D. Kandlur, D. Saha, and Kang G. Shin. BLUE: A new class of active queue management algorithms. Technical Report UM CSE-TR-387-99, University of Michigan, 1999.

[8] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1):1–15, 1994.

[9] S. Machiraju, M. Seshadri, and I. Stoica. A scalable and robust solution for bandwidth allocation. Technical Report TR UCB//CSD-02-1176, University of Califoirnia at Berkeley, 2002.

[10] S. Muthukrishnan. Data streams: Algorithms and applications, January 2003.

[11] Network simulator 2. Available at `http://www.isi.edu/nsnam/ns/`.

[12] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[13] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *SIGCOMM*, pages 118–130, 1998.

[14] Ion Stoica and Hui Zhang. Providing guaranteed services without per flow management. In *SIGCOMM*, pages 81–94, 1999.

[15] M. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

Appendix

Proof: (*Proof of Lemma 1*)

Let $\alpha = |A - B|, \beta = |B - A|$, and $\gamma = |A \cap B|$.

For Part (1) of the lemma, note that when π is chosen randomly from Π, the probability that the minimum of $\pi(B \cup A)$ falls in $B - A$ is at least $(1 - \delta)|B - A|/|A \cup B|$. Let $e = \beta/(\alpha + \gamma) > \epsilon$.

If $e < 1/2$, then

$$\frac{|B-A|}{|A \cup B|} = \frac{\beta}{\alpha + \beta + \gamma} \geq \frac{\beta}{(3/2)(\alpha + \gamma)} > (2/3)\epsilon.$$

If $e \geq 1/2$, then

$$\frac{|B-A|}{|A \cup B|} = \frac{\beta}{\alpha + \beta + \gamma} = \frac{1}{1 + \frac{\alpha+\gamma}{\beta}} \geq \frac{1}{3},$$

since $e \geq 1/2$ implies that $(\alpha + \gamma)/\beta \leq 2$.

For Part (2) of the lemma, we are given that $\alpha/(\alpha+\gamma) > \epsilon$ and $\beta/(\alpha + \beta + \gamma) < \epsilon$. When π is chosen randomly from Π, the probability that the minimum of $\pi(B \cup A)$ falls in $A - B$ is at least $(1 - \delta)|A - B|/|A \cup B|$.

$$\begin{aligned} \frac{|A-B|}{|A \cup B|} &= \frac{\alpha}{\alpha + \beta + \gamma} \\ &\geq \frac{\alpha}{\alpha + \gamma + (\frac{\epsilon}{1-\epsilon})(\alpha + \gamma)} \\ &= \frac{(1-\epsilon)\alpha}{\alpha + \gamma} \\ &> \epsilon(1 - \epsilon). \end{aligned}$$

■

Proof: (*Proof of Lemma 2*)

Fix $I \neq I' \in \mathcal{I}$. For any $j \in \{1, \ldots, k\}$, the probability that $h_j(I) = h_j(I')$ is exactly $1/2$. Therefore, the probability that $h_j(I) = h_j(I')$ for every j is exactly $1/2^k \leq \delta/n^2$, since $k \geq 2 \log n + \log \frac{1}{\delta}$. Summing over all $\binom{n}{2}$ pairs I, I' in $\mathcal{I}$, we have that the probability of any two items receiving the same signature is at most δ. ■