A Scalable and Robust Solution for Bandwidth Allocation

Sridhar Machiraju, Mukund Seshadri and Ion Stoica
University of California at Berkeley
Email: {machi, mukunds, istoica}@cs.berkeley.edu

Abstract

We propose a novel architecture for providing bandwidth allocation and reservation that is both scalable and robust. Scalability is achieved by not requiring routers to maintain per-flow state on either the data or control planes. To achieve robustness, we develop two key techniques. First, we use an admission control mechanism based on lightweight certificates and random sampling to prevent malicious users from claiming reservations that were never allocated to them. Second, we use a recursive monitoring algorithm to detect misbehaving flows that exceed their reservations. We randomly divide the traffic into large aggregates, and then compare the data arrival rate of each aggregate to its reservation. If an aggregate misbehaves, i.e., its arrival rate is greater than its reservation, we split and monitor that aggregate recursively until we detect the misbehaving flow(s). These misbehaving flows are then policed separately. We conduct extensive simulations to evaluate our solutions. The results show that the proposed solution is very effective in protecting well-behaved flows when the fraction of misbehaving flows is limited.

I. Introduction

The best-effort service model prevalent in the Internet allows routers to be stateless. However, as the Internet evolves into a commercial communications infrastructure, there is an increasing need for providing more sophisticated services such as bandwidth and delay guarantees. During the last decade a plethora of solutions have been developed to provide better services than best-effort. At one end, Integrated Services (IntServ) [6] are able to provide very powerful and flexible services. However, this comes at the cost of complexity. IntServ assumes a network architecture in which every router maintains per-flow state.

To alleviate the scalability concerns of IntServ, two new architectures have been proposed recently: Differentiated Services (DiffServ) [10] and Stateless Core (SCORE) [12][13]. These architectures differentiate between the edge and core routers in a trusted network domain, and achieve scalability by not requiring core routers to maintain any per-flow state, and by keeping per-flow state only at the edge routers. The reasoning behind this is that, in general, each edge router handles a smaller number of flows than a core router. However it is likely that the border routers routers carrying traffic between two large ISPs would have to handle a large number of flows *and* maintain per-flow state for all of them, since these routers lie on a trust boundary, and therefore will be required to function as edge routers. In addition, edge-based solutions are not robust since a single malicious or mis-configured edge router could potentially impact the whole domain.

Measurement-based admission control is another proposed approach to providing real-time services in the Internet [1]. In these designs, routers estimate the aggregate arrival traffic and make admission control decisions based on this estimate. These solutions are highly scalable, as routers do not need to maintain any per-flow state on both the data or control planes. Unfortunately, these solutions are not very robust. They offer little protection against malicious users that exceed their reservations, or that send traffic without making any reservation. Such misbehavior can easily compromise the quality of service of all users that share the same routers.

Thus, existing solutions to provide real-time services in the Internet suffer from scalability or robustness limitations. In this paper, we propose a novel solution that is *both* scalable and robust. In the absence of malicious users, our solution offers a service similar to the Premium Service [3] proposed in the context of DiffServ. In the presence of malicious users, our solution aims to protect well-behaved flows, by identifying and punishing *individual misbehaving flows*.

We achieve scalability by not requiring routers to maintain any per-flow state, even though our goal is to ensure that individual flows do not exceed their reservations. On the data plane, routers maintain a single queue for reserved traffic. On the control plane, routers maintain only the total reservation, which is used to perform admission control. To maintain an accurate estimate of the total reservation, end-hosts are required to periodically refresh their reservations.

While the solution described above is scalable, it is not robust. A malicious user or a mis-configured end-host that sends more traffic than its reservation permits can easily compromise the service of other users. Similarly, a malicious user that sends more/fewer refresh messages on the control plane than she is supposed to can affect the accuracy with which routers compute their total reservation.

In this paper, we introduce two key techniques toward realizing a robust and scalable QoS architecture. The first technique is the use of lightweight certificates (issued by routers) on the control plane. In conjunction with a soft state based control plane, certificates provide scalability by not requiring routers to maintain information about every individual flow. The strictly one-way nature of certificates along with the use of random sampling of refreshes achieves robustness.

The second technique we introduce is a stateless[1] *recursive monitoring* scheme for the data-plane that detects flows exceeding their reservations. This scheme works by dividing the traffic into large aggregates, and then estimating the arrival rate and the reservation of each aggregate. If an aggregate misbehaves i.e., its arrival rate is greater than its reservation, it is split and recursively monitored until the misbehaving flows are identified. We explore the parameter space of this algorithm in our proposed architecture and contrast it with a well known stateless solution - random sampling [4][14]. Our solution does not require routers to coordinate with each other; therefore a mis-configured router cannot adversely affect another router's behavior.

The rest of the paper is organized as follows. Section II provides the reader with background on QoS architectures. Section III defines our goals, service model and assumptions. Section IV describes our proposed control plane mechanism and Section V describes the data plane mechanisms. Section VII discusses additional issues and optimizations in our solutions. Section VIII presents simulation results and their implications. Section IX concludes, and outlines future work.

II. Background and Related Work

The IETF has proposed two major architectures for providing QoS in the Internet - Integrated Services (or IntServ) [6], and Differentiated Services (or DiffServ) [10]. IntServ provides strong service guarantees in terms of delay and bandwidth [11][15]. Control plane signaling (admission requests etc.) is done on a per-flow basis, and maintenance of per-flow reservation state at all routers is required. On the data plane, each router needs to perform per-flow management and maintain per-flow state for processing of data packets. Maintenance of large amounts of state prevents scalability of the network (in terms of the number of flows). In addition, robustness problems arise due to the difficulty of maintaining distributed replicated state in a consistent manner [7].

DiffServ distinguishes between edge and core routers. Edge routers are required to maintain per-flow state and process packets on a per-flow basis. However, core routers maintain state only at a coarse granularity and process packets based on Per-Hop-Behaviors (PHBs) corresponding to a field carried in the packet header, which is set by the edge routers. Thus, the data plane of the network core is scalable (in terms of number of flows). However the service provided is weaker, e.g. Assured Service and Premium Service [10].

In DiffServ, in order to address the issue of admission control (the control plane), the main proposal is to use a bandwidth broker, which acts as a centralized repository of information about the topology and the state of the network. Scalability may be achieved by either distributing or replicating the broker. This raises issues of fragmentation of resources and consistency of replicated information. Also, this method is appropriate only for long-lived flows with relatively rare flow setup and tear-down operations.

A third architecture [12][13] relies on Dynamic Packet State and provides stronger service guarantees than DiffServ, while maintaining per-flow state only at the edge routers. The edge routers encode the flow state into the packets themselves. This state decides the packet-processing behavior at core routers, which also modify this state.

The data plane algorithm proposed in this paper requires the maintenance of only aggregate state in the network. In our proposed algorithm, flows are randomly aggregated and such aggregate flows are policed in order to restrict the bandwidth of misbehaving flows without maintaining per-flow state. Aggregation techniques similar to the ones we use have been used earlier for use in congestion control algorithms. In particular, in the Stochastic Fair Blue [8] scheme, the authors use such a technique in order to rate-limit flows that do not respond to congestion signals given by the network.

A simple stateless detection technique that has been proposed in research literature is random sampling [4][14]. We contend that the recursive monitoring technique proposed in this paper can perform better than random sampling in many scenarios, and contrast the two algorithms.

III. Service Model, Goals and Assumptions

The solution proposed in this paper aims to provide "soft" bandwidth guarantees on a per-flow basis. A user can ask for a peak bandwidth allocation by sending a reservation request message. Once the reservation is granted, the user can send traffic at a rate *no* greater than the reserved bandwidth. In addition, for the entire duration of the reservation, the user has to periodically refresh the reservation. While we consider only the peak bandwidth allocation model in this paper, more sophisticated services such as the ones based on average-bandwidth or token-bucket specifications are possible. We intend to explore such services in the future.

The following are the important goals of our solution:

- **Service:** A flow that refreshes its reservation and does not exceed its allocated bandwidth, should experience a very low drop rate.
- **Robustness:** Routers should identify and isolate misbehaving flows in a timely fashion. The goal is to minimize the impact of misbehaving flows on well-behaved flows.

[1] We use the adjective "stateless" to refer to a solution which does not maintain state about every individual flow.

- **Scalability:** Routers should not be required to maintain per-flow state on either the data or control planes.
- **Deployability:** The solution should be incrementally deployable. Each router should operate independent of other routers and should make no assumptions about the neighbor routers implementing the same solutions.

In this paper we make two assumptions: (1) A *flow* is defined by the fields in the IP header. Without loss of generality, in this paper, we identify a flow using the IP source and destination addresses. (2) Routers can handle at least two types of traffic: best-effort, and priority traffic (for which our solutions are used). This is done by either using a Weighted-Fair Queuing scheduler [2], or a simple priority queue scheduler. We assume a priority bit in the packet header to differentiate between the priority and best effort packets. Finally, we make two points. First, the main goal on the control plane is to effectively protect the well-behaved flows, and not to achieve high resource utilization or low blocking probability which are important but secondary goals. Second, our solution does not provide absolute bandwidth or delay guarantees. This is because the detection of misbehaving flows is not instantaneous and it is possible for some well-behaved flows to experience significant drop rates.

IV. CONTROL PLANE

The main function of the control plane is to perform admission control. The challenge is to maintain accurate information about the total amount of reservation without maintaining per-flow state, in the presence of packet losses, delays and malicious flows that do not adhere to the signaling protocol. A secondary goal is to allow the data plane to compute the reservation of a given set of flows, as will be discussed in Section V.

A. Admission Control

We first discuss a simple signaling protocol that is able to handle control message delays and partial reservation failures. We later extend this signaling protocol to handle flows that misbehave on the control plane.

To perform bandwidth admission control, each router maintains the total amount of bandwidth reserved so far on an outgoing link, A. Upon receiving a reservation request for bandwidth r, the router simply checks whether $A + r \leq C$, where C is the link capacity. If this is true, the router accepts the reservation and updates A to be $A + r$. If all routers along the path admit the reservation, the reservation is granted, and the destination sends a reply to the source. Upon receiving this reply, the source can start sending data packets. Thus the latency incurred by the admission control process is one RTT.

The challenge is to maintain an accurate value of A in the face of partial admission failures, message losses and flow termination, without maintaining per-flow state. To address this challenge we use a soft-state approach similar to the one proposed in [12]. Each source that is granted a reservation is required to send periodic refresh messages as long as it intends to maintain the reservation. The interval between two successive refresh messages, denoted T_{ref}, is fixed and known by all routers that implement our solution. Each refresh message contains the bandwidth reserved by the flow. The router can then compute the aggregate reservation on the output link by simply adding the values carried in all the refresh messages received during a period T_{ref}. Since A is estimated based only on the control messages received during the most recent period, the scheme is robust to losses and partial reservation failures because inaccuracies do not build up over time.

A potential problem with the above solution is that consecutive refreshes of a flow can arrive more than a period T_{ref} apart due to delay jitter. This may cause the control plane to "miss" the flow when it computes the aggregate reservation. One way to alleviate this problem is to compute the aggregate reservation over a larger period of time. We call such a period a *router period* and denote it by T_{router}. For simplicity, we choose T_{router} to be a multiple of T_{ref}, that is, $T_{router} = n_{ref} * T_{ref}$. We also assume that the maximum jitter does not exceed T_{ref}, so that at most 1 refresh is "missed". We then overestimate the aggregate reservation by a factor f, to account for the "missed" refreshes. The computation of f is presented in Section IV-A.1. Such over-provisioning can also be employed to account for refresh message losses, if the loss probability is known.

Further, when a router receives (and accepts) a reservation request for bandwidth r in the *middle* of a router period, (at time d from the beginning of the router period), then the router increments its total reservation estimate A by $r \times \lceil d/T_{ref} \rceil$, not just by r.

1) Over-provisioning Computation: We now demonstrate the computation of the over-provisioning factor f mentioned earlier. We assume a bounded delay jitter $q \leq \frac{T_{ref}}{2}$. In addition, we assume that refresh messages are transmitted periodically every T_{ref} seconds. Next, we compute a bound on the number of refreshes received during a router period (T_{router}).

With n flows in the system, ideally we expect to receive $n \cdot n_{ref}$ refreshes during each router period. However, due to the delay jitter, some of the refreshes may not be received in the current router period. The probability that only $n_{ref} - 1$ refreshes of a flow are received during the router period is $p = \frac{q}{T_{ref}}$. Let N_{ref} be the number of refreshes received during a router period. The probability that more than u refreshes are *not received* in the current router period due to delay jitter is $\varepsilon = \Sigma_{i=u+1}^{i=n} C_i^n \cdot p^i \cdot (1 - p)^{n-i}$. By using the Chernoff bound to approximate the sum, the total amount of reservation on the link is less than $f \cdot N_{ref}$

(where f is given by $\frac{n_{ref} \cdot n}{n_{ref} \cdot n - u}$) with a probability greater than $1 - \varepsilon$. Thus, the admission control test for a reservation request r becomes

$$f \cdot A + r \cdot n_{ref} \leq C. \tag{1}$$

where C is the link capacity.

B. Control Plane Misbehavior

The reservation estimation algorithm presented above assumes that all sources obey the control plane protocol. In this paper, we focus on the following three ways in which a source can misbehave on the control plane: (1) Sending refresh messages without having been admitted. (2) Stop sending periodic refreshes and resume sending them later without undergoing admission control again. (3) Sending more than n_{ref} refresh messages in a router period.

Scenario (3) is addressed in Section IV-C. Scenarios (1) and (2) can result in resource under-provisioning, which can compromise the service of well-behaved flows. Note that the stoppage of refreshes in scenario (2) is equivalent to flow termination as far as the router's control plane is concerned. Therefore when the flow resumes sending refreshes, its refreshes will correspond to non-existent reservations. How does the router detect that without per-flow state? We now present our solution to address scenarios (1) and (2).

The key idea is to use lightweight certificates generated by routers in order to allow the routers to verify that a refresh message corresponds to a flow which was admitted earlier. Each router along the path computes and attaches a certificate to the admission request message if the admission succeeds. The request message collects the certificates from all routers along the path, and the destination sends them back to the source in a response message. Subsequently, the source sends refresh messages containing all the certificates that it had received, in the same order that they were received.

The certificate issued by a router is a one-way hash of the requested reservation amount, the flow identifier, and a router-specific key (see Fig.1). Each router on the path (which implements our solution) needs to know only its own key value, and how to access the certificate issued by it (earlier) from the control packet payload. We refer to the fields in which these certificates are stored, in the refresh packet, as *C-Fields*.

Whenever a router receives a refresh message, the router re-computes a certificate C_1 using the packet fields and its key. It then retrieves the certificate C_2, which was issued by it during admission control, from the appropriate C-Field in the refresh packet. If C_1 and C_2 are identical the router is able to conclude that (a) the refresh indeed corresponds to an earlier admitted flow, and (b) the reservation amount specified is the same as requested during admission time. In other words the certificate is "valid". Therefore the router accepts the refresh i.e., uses the specified reservation amount to update A.

As an optimization, the refresh message also contains an *Offset* field which indicates how many routers (which implement our solution) the message has traversed. This field is incremented every time the message traverses a router, and can be used by routers to efficiently access their certificate.

This solution addresses scenario (1). However, a source that was granted a reservation and stopped sending refreshes and data (and was thus inferred to have terminated), can use the certificates at a later time without getting readmitted, which would defeat our scheme. To address this problem, routers change their keys, and therefore the certificates for each flow, during every refresh period (T_{ref}). The certificates in the arriving refresh packet are replaced by the new certificates. The refresh packet reaches the destination which sends back the payload to the source. Thus routers can convey their new certificates to the sources. Sources are expected to use the new certificates in the next period. Thus, if a source stops sending refreshes it will stop receiving the new certificates. If it starts sending refreshes later, they will not contain valid certificates. There remains the possibility of lost or delayed refreshes causing well-behaved flows to not receive the new certificates. We address this issue in Section VII.

The control plane operations described here are shown as pseudo-code in Fig.2. The function **isValidCert()** checks if the certificate in the packet's appropriate C-Field matches the certificate computed from the current key and the header fields. If it is valid and is not an "extra refresh" (explained in Section IV-C), the router updates its estimate of total reservation, in **updateResvEstimate()**. If the certificate is not valid, then the refresh message is treated simply as a new admission request message (in **admissionCheck()**).

C. Detection of Extra Refreshes

A flow F may misbehave by sending more than one refresh message every T_{ref} seconds. In particular, if F sends $x_{ref} > 1$ refresh messages every T_{ref} seconds , it can "inflate" its reservation by a factor x_{ref}, *without undergoing admission control* for that amount. This is because the aggregate reservation estimation algorithm will add up the reservations carried by all refresh messages during a T_{ref} period. This can prevent other flows from being admitted! In addition, the flow could send x_{ref} times the amount of data it was admitted for, and the data plane will not detect this occurrence, since the flow does not actually exceed its

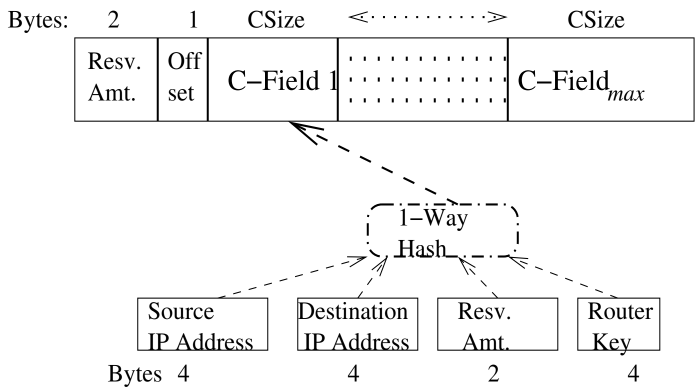

Fig. 1. Control Packet Format

```
processControlPacket(Packet p)
  fid = getfid(p);
  cert = getCertificate(p);
  //processing a valid refresh
  if isValidCert(cert)
    if isExtraRefresh(fid)
      //downgrade/contain the flow
      barFlow(fid);
    else
      updateResvEstimate(p);
      updateDataPlane(p);
  else //admission request
    if admissionCheck(p)
      attachCertificate(p);
    else
      denyAdmission(p);
```

Fig. 2. Pseudo-code: processing control packets

```
isExtraRefresh(Packet p)
  fid = getfid(p);
  hashval = hash(fid);
  sample = sample[hashval];
  if isSampleEmpty(sample) or
    finishedPolicing(sample);
    //replace with probability q
    replaceSample(sample, fid, q);
  if sampleHasFid(sample, fid)
    //increment the refresh count
    updateSample(sample, p);
    if isMisbehaving(sample)
      //downgrade/contain the flow
      barFlow(fid);
```

Fig. 3. Pseudo-code: Random Sampling

(inflated) reservation. Therefore, the challenge is to detect such “extra” refreshes without per-flow state. We use random sampling to address this problem.

Random sampling is a well known method to monitor flows that has been proposed earlier [4][14]. The basic algorithm is to choose a packet randomly and monitor the flow to which the packet belongs to. We now describe a random sampling scheme to detect flows that send extra refreshes.

If exactly one flow can be monitored at any given time, this flow could be chosen to be the originator of a refresh randomly chosen (i.e., sampled) from the stream of refreshes arriving at the router. Note that the probability of choosing a particular flow increases linearly with the number of refreshes the flow sends in a period T_{ref}. Also, flows that misbehave more (and waste more resources) are likely to be detected before flows that misbehave less. Once a flow is chosen, it is monitored for a period T_{mon} and then classified as well-behaved or misbehaving. Then it is replaced by the flow corresponding to another randomly sampled refresh, and so on. T_{mon} can be equal to T_{ref}: however there is a risk that 2 successive refreshes of a well-behaved flow arrive sooner than T_{ref}, due to network jitter effects. Therefore T_{mon} could be chosen to be a multiple of T_{ref} in order to use a longer observation period.

We now make two generalizations to this method. First, when the monitoring of a flow is finished, another flow (corresponding to an arriving refresh) can be chosen to be monitored with a probability of q (as opposed to probability 1). Second, if B flows can be monitored simultaneously ($B > 1$), we can first use a hash function for mapping flow identifiers (of arriving refreshes) onto an index in the range $[1..B]$, and then choose a flow to be monitored only if no other flow mapping onto that same index is being currently monitored. The complete algorithm is depicted in Fig.3, in the routine for **isExtraRefresh()**. This is the control plane function which determines, upon receiving a refresh, whether that flow is sending more refreshes than it should.

We observe at this point that this method can be adapted for use on the data plane, as an alternative to our proposed recursive monitoring algorithm, to detect flows which use more bandwidth than they reserved, by sampling data packets instead of refresh packets.

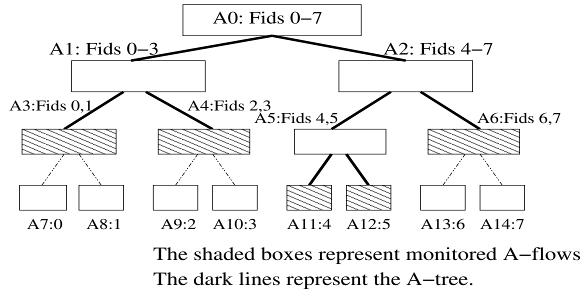

Fig. 4. Illustration of an *A-Tree*

V. DATA PLANE

The goal of the data plane is to ensure that each flow receives its reserved bandwidth. The challenge is to achieve this in the presence of misbehaving flows that exceed their reservations, without maintaining per-flow state.

To address this challenge, we employ a monitoring mechanism for detecting misbehaving flows. Once detected, a misbehaving flow is contained or downgraded. A simple approach to detect misbehaving flows is to monitor a (bounded-size) set of flows: if a monitored flow's data rate exceeds its reservation, the flow is assumed to misbehave. The key question is: how do we select the set of flows to be monitored? One possible solution is to select these flows *randomly* (as in Section IV and [14]). However, we propose a more elaborate scheme, called *recursive monitoring* (abbreviated to *R-Mon*), that (as shown by the simulation results in Section VIII) is able to out-perform the random sampling scheme in many scenarios. We now describe our *R-Mon* algorithm.

A. R-Mon Algorithm Overview

The basic idea of *R-Mon* is to *randomly* divide the total set of flows into large aggregates and monitor the aggregates. When an aggregate misbehaves, it is recursively split into smaller aggregates which are monitored similarly. The recursion terminates when the monitored aggregate consists of a single flow. Section V-B describes how this algorithm works when the total number of aggregates that can be simultaneously monitored is bounded. This bound represents the space complexity of our algorithm.

An important advantage of our algorithm is that it gives a deterministic bound on the time it takes to catch a misbehaving flow assuming that the misbehavior is significant enough to cause losses to well-behaved flows. On the other hand, our algorithm cannot catch a flow faster than the time it takes to expand a misbehaving aggregate into its constituent flows. In the worst case, this time is proportional to $\Omega(\log m)$, where m is the number of flows in the aggregate. In contrast, the random sampling scheme can select and catch the misbehaving flow right away, but it cannot provide a deterministic bound on how long it takes to catch a flow. The trade-offs between these two algorithms are explored experimentally in Section VIII and analytically in Section VI.

B. The A-Tree

We now describe the data structures used to monitor and expand misbehaving aggregates. Consider a router which intends to monitor the traffic on a single link. Let an *A-flow* (short for "Aggregate-flow") be an arbitrary subset of flows traversing that link. Our scheme is based on a simple observation: *if the traffic of an A-flow exceeds the total reservation allocated to all flows in the A-flow, then at least one of these flows is misbehaving.*

It is useful to think of A-flows as nodes in a tree (see Fig.4). Consider a *complete* tree whose root represents the A-flow consisting of all flows. The leaves of this tree represent A-flows each containing a single flow traversing the link. Each internal node represents an A-flow which also has a set of "children" A-flows which are pairwise disjoint and whose union is the parent A-flow. The number of nodes of this complete tree that can be maintained at any given time is constrained by the amount of state the router can maintain. Let B be the maximum number of A-flows that can be simultaneously monitored by the router. The sub-tree which has as its leaves the A-flows that are currently being monitored is called the *A-tree*.

The central idea of *R-Mon* is to recursively descend down those branches of the A-tree that lead to misbehaving flows. The pseudo-code for this is shown in Fig.5. To implement this scheme, the router maintains two data structures:

- *monTbl*, which maintains the set of A-flows and flows that are currently being monitored. The size of this set is bounded by B.
- *alertList*, which is a priority queue that maintains the set of A-flows that are misbehaving. These represent the set of nodes in the *A-Tree* that need to be expanded/contained in the future. To decide which one of these A-flows to expand first (i.e., a priority function), we use a simple heuristic: given two A-flows at different levels, we first expand the A-flow which has a higher depth in the *A-Tree*; given two A-flows at the same level we expand the one that misbehaves by a larger amount.

```
processDataPacket(Packet p)
   fid = getFlowId(p); //compute flow ID from p's header
   updateAFlows(monTbl, fid, p); //update resource consumption


/* this is run every T_ref seconds */
periodicSweep()
   //create priority list of alerts
   alertList = sweepAndCreateAlertList(monTbl);
   while notEmpty(alertList)
      aflow = removeTopPriorityAFlow(alertList);
      if isSingleFlow(aflow) //A-flow is a single flow
         barFlow(aflow.flowId); //contain/downgrade flow
      else //remove lowest priority A-flows
         makeSpaceForExpansion(monTbl, aflow);
         insertChildAFlows(monTbl, aflow); //split A-flow
      removeFromTable(monTbl, aflow);
```

Fig. 5. Pseudo-code: processing data packets.

In the absence of misbehaving flows, the algorithm tries to evenly extend the complete *A-Tree* as deep as allowed by the bound B, i.e., up to a depth $\log_k^B$, where k is the degree of the *A-Tree*. When one of the A-flows misbehaves, its children are added to *monTbl*. In order to add these children, we remove all A-flows which have been monitored and have not been observed to be misbehaving. If space constraints remain, we might need to remove A-flows (of lower priority) from *monTbl*. Since the removal of A-flows could result in some flows not belonging to any monitored A-flow, in practice, we always monitor every A-flow at depth one (e.g., $A1$ and $A2$ in Fig.4). Note that depth zero corresponds to the root of the A-Tree, for which state need not be actually maintained.

We now examine the issue of determining the aggregate reservation of each A-flow in *monTbl*. At this point we make an assumption that the control plane informs the data plane of the arrival of these refresh messages; this is the only information that has to be exchanged between the control and data planes. The data plane uses the period T_{ref} to count the number of refresh messages[2] received by each monitored A-flow and determines the aggregate reservation of that A-flow. At the end of each such period, the router does the following for each A-flow in *monTbl*: compare the total traffic sent in that period by that A-flow with its aggregate reservation (bandwidth); if the A-flow is found to be misbehaving, then either its children are added to *monTbl*, or if the A-flow consists of a single flow, that flow is downgraded or contained.

Fig.5 shows the pseudo-code for the data plane operations. Each data packet is handled by **processDataPacket()** which simply updates the resource consumption of every A-flow to which the packet's flow belongs to. Every T_{ref} seconds, **periodicSweep()** is run, which removes well-behaved A-flows, creates the *alertList*, and either downgrades/contains/bars (**barFlow()**) or splits and expands (**insertChildAFlows()**) the A-flows in the *alertList*.

C. Discussion

The data plane mechanism makes the assumption that exactly one refresh per flow is received in each period T_{ref}. However, in a real network, refreshes may arrive earlier, be delayed or lost. In the case when the number of refreshes received is smaller than the number of flows (in an A-flow), the A-flow may be (wrongly) classified as misbehaving. There are at least two solutions to this problem. The first solution is to compute the reservation of an A-flow by taking into account refresh message delays and losses. The second solution is to compute the reservation without taking delays and losses into account. Note that in the latter case, if the A-flow is a single flow, we know that it is supposed to send 1 refresh in the last T_{ref} seconds. If that refresh doesn't arrive, we know for certain that we have not received accurate reservation information for that flow, and will not bar that flow (we can choose to monitor it for another period). Therefore there is no danger of mis-classifying a well-behaved flow as a misbehaving flow. The only possible penalty is inefficiency, not correctness. We have chosen the second method for the simulations in this paper.

The algorithm described allows (but does not require) the containment of A-flows while they are being monitored. In this way, we can bound the total traffic sent by a flow by the reservation of the deepest A-flow in *monTbl* which contains that flow. In our implementation, we chose to not contain A-flows. This decision is motivated by the fact that containing an A-flow may adversely affect the well-behaved flows within that A-flow even when there are enough resources in the system to handle all the flows.

[2]Recall that each flow is supposed to periodically send refresh messages with period T_{ref}.

One remaining question is how to map flows to A-flows. One simple way to achieve this is to use a subset of bits from the flow identifier[3] to determine the A-flow to which the flow belongs to. For instance, a 32-bit flow identifier could be used to construct an A-tree of maximum depth 16 and degree 4 by associating 2-bit masks with each level. In particular, at the first level of the *A-Tree* there can be at most four A-flows, each of which consists of all flows with a prefix of $00, 01, 10$ or 11.

Once an individual flow is classified as misbehaving, we assume a containment procedure that either downgrades the flow's traffic to best-effort or shapes it to its reserved bandwidth (referred to as **barFlow()** in Fig.5). One way to achieve this is by maintaining per-flow state (either at each router, or only at the edge routers of a domain) for each flow that has been detected to be misbehaving. This state could be timed out in order to allow for flows which resume correct behavior. We note that this containment procedure imposes a lower limit on the amount of state required at the routers; our solutions are inherently suitable only for scenarios in which the number of misbehaving flows is small (though they might misbehave by large amounts). If we were to follow the approach outlined above, it would be better to shape the detected flows to their reservations than to downgrade them, in order to account for the possibility of flows wrongly detected to be misbehaving. While the detailed design of the flow containment mechanism is beyond the scope of this paper, in Section VII we describe an alternative stateless scheme to implement flow containment.

One important point to note is that if an A-flow is well-behaved, this does not necessarily mean that all flows in that A-flow are well-behaved. A misbehaving flow can "hide" behind other flows (in the same A-flow) that use less bandwidth than they reserve. This is acceptable since our main goal is to protect well-behaved flows and not catch all misbehaving flows.

VI. Analysis

In this section we analytically explore trade-offs in the data plane algorithms, involving the amount of state maintained, time taken to detect misbehaving flows, and other system parameters.

A. Static Scenario

This section presents an analysis of both data plane algorithms in a "static" system with a single link governed by a router. This system begins with a certain number of flows in the system (which desire to send data through that link). A certain fraction of those flows are misbehaving flows. No subsequent flows arrive into the system. We calculate the maximum time taken to detect all misbehaving flows in such a system. Later, we present an analysis of a "dynamic" system i.e., one in which flows continuously enter and leave the system.

Our first theorem illustrates the trade-off between the state maintained by the *R-Mon* algorithm in the router, and the time the router takes to detect all the misbehaving flows in a static system. As expected, the time taken to detect all the misbehaving flows increases with their number (M), and decreases with the maximum number of A-flows or flows that can be simultaneously monitored (B).

Theorem 1: Assume that there are M misbehaving flows out of a total of N flows in a static system at time 0, and each misbehaving flow causes every A-flow to which it belongs to misbehave. Let B be the maximum number of A-flows that can be monitored simultaneously, using the *R-Mon* algorithm. B_s A-flows at the top of the A-Tree are always maintained, and up to B_d more A-flows can be dynamically added to the A-Tree (note that $B = B_s + B_d$). Let k be the degree of the A-tree. Then, the time to detect and contain the M misbehaving flows using an optimal recursive monitoring scheme is bounded by

$$T_{ref} \cdot \log_k \frac{N}{B_s} \quad if\ M \le B_d/k,\ and$$

$$T_{ref} \cdot \left(\log_k \frac{B_d}{B_s} + \frac{Mk}{B_d} \cdot \left(\log_k \left(\frac{N}{M} \right) + \frac{1}{k-1} \right) \right) \quad otherwise.$$

Proof: For simplicity, assume that both B and M are powers of k. If there are M misbehaving flows, the resulting A-tree has at most $M-1$ branching points (a branching point in the A-tree corresponds to a misbehaving A-flow that is split into k misbehaving A-flows).

The B_s A-flows that are always maintained correspond to depth d_s. Let $m(d)$ (defined for $d \ge d_s$) be the maximum possible number of nodes representing *misbehaving* A-flows at depth d in the A-tree. $m(d_s)$ is the minimum of B_s and M. Since for each misbehaving A-flow, the router monitors up to $k-1$ well-behaved A-flows, the total number of A-flows that need to be monitored at depth d is bounded by $m(d) \times k$. If $M \le B_d/k$, then all M misbehaving flows can be detected in $\log_k N/B_s$ refresh periods (T_{ref}). If not, let d_1 be the smallest value of d such that $m(d) > B_d/k$. In the worst case, starting with depth d_1 the router can

[3] The flow identifier can be computed by concatenating or hashing the flow's source and destination IP addresses and eventually port numbers.

no longer monitor all A-flows at each depth at the same time. The total number of nodes (b) in the A-Tree at depths equal to and greater than d_1 is given by,

$$b \leq \sum_{i=d_1}^{d_{max}} m(i) \ \times k.$$

where $d_{max} = \log_k B_s$. Then the number of refresh periods it takes to verify all A-flows that correspond to these b nodes is b/B_d. Since $m(i) \leq M$, it follows that b is maximized when $m(i)$ reaches M as fast as possible. Since the minimum depth at which $m(i)$ can reach M is $d_1 + l$, where $l = \lfloor \log_k(M/m(d_1) \rfloor + 1$, it follows that the maximum value of b is

$$k \cdot \frac{Mk - m(d_1)}{k-1} + \sum_{i=d_1+l+1}^{d_{max}} Mk.$$

Thus the total number of periods to detect all misbehaving flows, $d_1 + b/B_d$, is bounded by

$$d_1 + \frac{1}{B_d}\left(k \cdot \frac{Mk - m(d_1)}{k-1} + Mk\left(d_{max} - d_1 - \log_k \frac{M}{m(d_1)} - 1\right)\right).$$

It is easy to see that the above expression is maximized when d_1 is minimized, and $m(d_1)$ is maximized. Since $m(d_1) \geq B_d/k$, and since it takes at least $d_1 = \log_k(B_d/B_s)$ refresh periods to reach the maximum $m(d_1)$(which is B_d), it follows that the router can detect all misbehaving flows after at most

$$\log_k \frac{B_d}{B_s} + \frac{Mk}{B_d}\left(\frac{k - \frac{B_d}{Mk}}{k-1} - 1 + \log_k\left(\frac{N}{M}\right)\right) \leq \log_k \frac{B_d}{B_s} + \frac{Mk}{B_d} \cdot \left(\log_k\left(\frac{N}{M}\right) + \frac{1}{k-1}\right).$$

periods. □

The next theorem shows how the mean time taken to detect all the misbehaving flows in a static system decreases with increasing values of B and the degree of misbehavior (α). This behavior is in contrast with that of the *R-Mon* scheme which is independent of the degree of misbehavior.

Theorem 2: In a static scenario, if a router uses random sampling of flows on the data plane, then the average time taken to detect 99% of the misbehaving flows, which send data at α times the rate permitted by their reservations, is given by

$$t = \frac{T_{ref}}{B}\left(0.99 \cdot M + \frac{N-M}{\alpha} 4.6\right)$$

Proof: Let m denote the expected instantaneous number of misbehaving flows. This is a monotonically non-increasing function of time t, defined when t is a multiple of T_{ref}, and is described by

$$\Delta m = -\frac{\alpha m}{\alpha m + N - M} \cdot B \cdot \frac{\Delta T}{T_{ref}}$$

The above equation could be approximated by the differential equation

$$dm + \frac{N-M}{\alpha m} dm = -\frac{B}{T_{ref}} \cdot dt$$

Solving the above equation, we get

$$m + \frac{N-M}{\alpha} \ln\ m = M + \frac{N-M}{\alpha} \ln\ M - \frac{B}{T_{ref}} t$$

$$t = \frac{T_{ref}}{B}\left((M-m) + \frac{N-M}{\alpha} ln\, \frac{M}{m}\right).$$

□

B. Dynamic Scenario - A General Result

We now begin the analysis of the random sampling and *R-Mon* algorithms in the context of a dynamic system, in which flows arrive and depart (terminate) all the time. We prove a general result about the fraction of misbehaving flows remaining in a stable system in the presence of a policing element.

Theorem 3: Consider a system in which flows arrive at the rate λ. Let f_m be the fraction of incoming flows that misbehave. Let L be the average lifetime of these flows. The system is policed by a mechanism which removes misbehaving flows at a rate of r ($r \leq f_m \cdot \lambda$). If the fraction of misbehaving flows in the system converges to an equilibrium value f_e, then

$$f_e = \frac{\lambda \cdot f_m - r}{\lambda - r}$$

Proof: If f_e converges to an equilibrium, then let M_e be the number of misbehaving flows in the system at this equilibrium. Let N_e be the number of flows in such an equilibrium system. N_e consists of both well-behaved flows and misbehaving flows. Using Little's theorem we obtain the number of well-behaved flows (at this equilibrium) G_e, given by $\lambda \cdot (1 - f_m) \cdot L$.

Misbehaving flows arrive at a rate of $f_m \cdot \lambda$ in the system. The element removes r flows per unit time, on average. Hence, applying Little's theorem,

$$M_e = L \cdot (\lambda \cdot f_m - r)$$

The equilibrium fraction of misbehaving flows f_e, is given by $M_e/(M_e + G_e)$ i.e.,

$$f_e = \frac{\lambda \cdot f_m - r}{\lambda - r}$$

□

C. Control Plane

Theorem 4: Consider a dynamic system with parameters as described in Section 3, in which misbehaving flows misbehave only on the control plane, by sending β refreshes instead of 1 ($\beta > 1$) per refresh period (T_{ref}). Assume that a random sampling mechanism, as described in Section IV-C, is employed to detect and remove misbehaving flows from the system. Also assume that the random sampling mechanism monitors flows for a period of T_{ref}. If the fraction of misbehaving flows remaining in this system converges to an equilibrium value f_e, then f_e satisfies

$$f_e^2 \cdot (\beta - 1 - \frac{B \cdot \beta}{\lambda \cdot T_{ref}}) + f_e \cdot (1 + \frac{B \cdot \beta}{\lambda \cdot T_{ref}} - f_m \cdot (\beta - 1)) - f_m = 0$$

Proof: In this case, the number of flows detected and removed per period T_{ref} by the random sampling mechanism is given by multiplying the probability that a flow chosen to be monitored is a misbehaving flow (p_m), with the number of flows that can be monitored simultaneously (B). In other words,

$$r = \frac{B \cdot p_m}{T_{ref}}$$

Also, p_m is the fraction of refreshes sent by misbehaving flows i.e., $p_m = \frac{\beta \cdot f_e}{(\beta-1)f_e+1}$. Substituting for r in Theorem 3, we obtain the desired result. □

Theorem 5: Consider the system described in Theorem 4. Assume that the refreshes of a flow are sent at uniform intervals. Also assume that a flow is monitored until it misbehaves, subject to a maximum time T_{ref} (note that this assumption differentiates this theorem from Theorem 4). If the fraction of misbehaving flows remaining in this system converges to an equilibrium value f_e, then f_e satisfies

$$f_e^2 \cdot \frac{B\beta}{\lambda T_{ref}} - f_e \cdot (1 + \frac{B\beta}{\lambda T_{ref}}) + f_m = 0$$

Proof: In this case, $r = \frac{B \cdot p_m}{T}$ where p_m is as in Theorem 4 and T is the average time taken to monitor a flow (which might be either well-behaved or misbehaving).

$$T = (1 - p_m)T_{ref} + p_m \frac{T_{ref}}{\beta} = \frac{T_{ref}}{1 + (\beta - 1)f_e}$$

Hence $r = \frac{B \cdot \beta \cdot f_e}{T_{ref}}$. Substituting for r in Theorem 3, we obtain the desired result. □

D. Data Plane

We now consider a "dynamic" system that has flows arriving and leaving the system at a steady rate, in which misbehaving flows misbehave on the data plane.

Theorem 6: Consider a dynamic system with parameters as described in Section 3, in which misbehaving flows misbehave only on the data plane, by sending data at α times the rate permitted by their reservations. Then, for an optimal recursive monitoring scheme used to remove misbehaving flows at rate r, in which $B_s = k$ (both terms as defined in Theorem 1), the value of r is maximum when

$$k = e^{\frac{(l-\sqrt{l^2-4l})}{2}} \quad where\ l = \ln\ N.$$

Proof: Let k, B_s, B_d, T_{ref},N be as defined as in Theorem 1 and λ, L, r, f_m as in Theorem 3. For an optimal recursive monitoring scheme, for an average of r flows to be barred in a period, up to $r \cdot k$ A-flows need to be maintained at each level of the A-tree. Then, if d is the depth of the A-Tree, B_d must be at least $r \cdot k \cdot d$. Note that $d = \log_k \frac{N}{B_s}$. Hence,

$$r = \frac{B_d}{k \cdot d \cdot T_{ref}} = \frac{B_d}{k \cdot \log_k \frac{N}{B_s} \cdot T_{ref}}$$

The optimal value of the degree k is one which maximizes r. If B_s equals k (as in all our simulations), the best value of k is one which minimizes $k \log_k \frac{N}{k}$. It can be easily shown that this expression is identical to $(k \cdot \frac{\ln N}{\ln k} - k)$. This function reaches its minimum at $k = e^{\frac{(l-\sqrt{l^2-4l})}{2}}$ where l is $\ln\ N$. Note that we can substitute for r in Theorem 3 to obtain a specific result about f_e. □

Figure 7 shows the variation of the optimal degree with varying number of flows. This graph indicates that the optimal degree is 3.4 for $n = 1000$. This matches our simulation scenarios with $n = 1000$, whose results show that degree 4 is optimal.

Theorem 7: Consider a dynamic system with parameters as described in Section 3, in which misbehaving flows misbehave only on the data plane, by sending data at α times the rate permitted by their reservations. Assume that a random sampling scheme, which can monitor at most B flows simultaneously and monitors flows for a period of T_{ref}, is employed to detect and remove misbehaving flows from the system. If the fraction of misbehaving flows remaining in this system converges to an equilibrium value f_e, then f_e satisfies

$$f_e^2 \cdot \frac{B\alpha}{\lambda T_{ref}} - f_e \cdot (1 + \frac{B\alpha}{\lambda T_{ref}}) + f_m = 0$$

Proof: This is proved in the same way as Theorem 5, with β being substituted by α. □

E. A Comparison

Consider a dynamic system with data plane misbehavior, being policed by either the random sampling scheme or the recursive monitoring scheme. If the fraction of misbehaving flows remaining in this system converges to an equilibrium value f_e, and if $r = f_m \cdot \lambda$, then clearly f_e is negligible. We now examine the minimum value of B required to meet this condition, for both the random sampling and the recursive monitoring algorithms,

We first consider the recursive monitoring scheme. Assuming that $B_s = k$ and L is the average flow lifetime, f_e is negligible if

$$f_m \leq \frac{(B-k) \cdot L}{k \cdot (\log_k\ N - 1) \cdot N \cdot T_{ref}} \Rightarrow (N \cdot \ln\ N) \left(\frac{k \cdot T_{ref} \cdot f_m}{L \cdot \ln\ k} \right) \leq B$$

We now consider the case where the random sampling scheme is employed. Substituting β by α in Theorem 5 (if misbehaving flows send data at α times the reserved rate), we obtain the result that f_e is less than $\frac{f_m}{z}$ (for $z > 1$), when

$$z \leq \frac{B \cdot \alpha \cdot L}{N \cdot T_{ref}} \Rightarrow N \left(\frac{z \cdot T_{ref}}{L \cdot \alpha} \right) \leq B$$

Thus, random sampling performs better (i.e., requires a lower value of B for the same value of $f_e = \frac{f_m}{z}$) than recursive monitoring (with $k = 4$) when

$$(N \cdot \ln\ N) \left(\frac{k \cdot T_{ref} \cdot f_m}{L \cdot \ln\ k} \right) \geq N \left(\frac{z \cdot T_{ref}}{L \cdot \alpha} \right)$$

$$\Rightarrow \log_{10} N \;\geq\; \frac{0.15 \cdot z}{\alpha \cdot f_m} \;\; i.e.,\; N \;\geq\; 10^{\frac{const}{mis}}$$

where $mis = \alpha \cdot f_m$ and $const$ is a constant value. The variable mis can be considered to be a measure of the total "amount" of misbehavior. This result implies that as the total "amount" of misbehavior increases, random sampling performs better than *R-Mon*, but in scenarios where the total amount of misbehavior is low *R-Mon* performs better.

VII. Other Issues

In this section we examine certain important issues and possible optimizations; though these are not present in our simulation scenarios, they are required for a practical deployment of our solutions.

Route changes: If a flow's route changes, some routers on the new path will no longer find a valid certificate in the refresh. Therefore, they will interpret this as an admission request. This is acceptable behavior because, once the path changes, it is necessary to allocate network resources on the new path. Meanwhile, the flow's reservation with routers on the old path which are not in the new path, or not in the same position in the new path, will be automatically timed out, due to the soft state reservation approach.

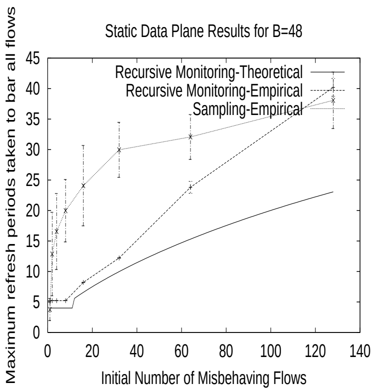

Fig. 6. A typical result in the static case

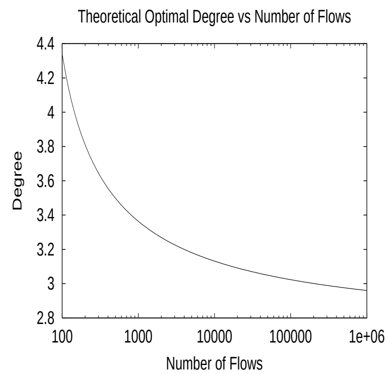

Fig. 7. Theoretical Optimal Degree vs n

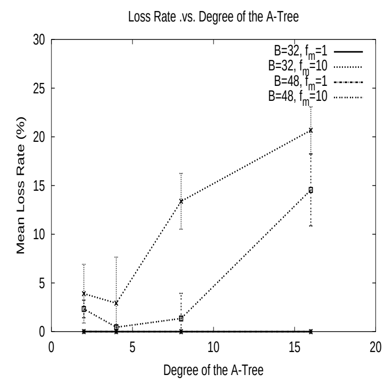

Fig. 8. Experimental Loss Rates with Varying Degree

Sources of Overhead: T_{ref} must be chosen so that the bandwidth taken by refresh messages is low compared to the data rate. However, there is a trade-off between T_{ref} and the QoS received by well-behaved flows (since misbehavior detection will take longer with larger T_{ref}). Assuming an C-Field size of 16 bytes (CSize in Fig.1), a refresh of size 500 bytes would suffice since most Internet paths contain less than 30 hops. Considering an example scenario from our simulations, an 80 Kbps data rate flow with a refresh period of 5 seconds has a bandwidth overhead of 2%. Another source of overhead is the processing of each control and data packet by routers.These operations are likely to be fast since they are all based on use hash operations (including certificate creation/verification). However, this overhead needs to be investigated in an implementation, particularly since a high certificate processing overhead would make the router vulnerable to Denial of Service attacks based on flooding the router with packets containing invalid certificates.

Deployment Issues: Our solution does not require any information exchange among routers. The only coordination required is to fix the value of T_{ref} to a well-known value. In addition, even partially deploying our solution to the known congested routers would significantly improve the level of service in the network (see Sec. VIII). Thus our solution can be incrementally deployed.

Message Losses: The possibility of message losses exists in real networks. If a refresh message is lost, the system would detect a well-behaved flow as misbehaving on the data plane. This problem may be overcome by over-provisioning the data plane based on the expected loss rate. In addition, referring to Fig.2, a flow whose refresh is lost is forced to undergo admission control. A possibility for optimization exists here: if the refresh does not cause over-reservation, the router can attach a new certificate to the refresh packet, and actually refresh the reservation, even if the refresh packet's certificate is not valid. In order to prevent a misbehaving end-host from taking advantage of this, such action may be "recorded" in the certificates. A flow with several late "recorded" refreshes could be deemed to be misbehaving.

Stateless Barring of Flows: The use of certificates also allows us to avoid maintaining state for flows that have already been classified as misbehaving. The certificates returned to such flows could be computed using a different router key than the one used for well-behaved flows. This solution requires that *each data packet* carry such a certificate. A router can detect whether a data packet belongs to a misbehaving or an well-behaved flow by simply checking its certificate. If the packet belongs to a misbehaving flow, the router can downgrade the packet to the best-effort service. While such a solution can completely eliminate the need of maintaining state for misbehaving flows, there are two potential problems. First, per-data-packet overhead increases now as a router needs to compute certificates for every data packet (though the hashing operation involved can be efficiently implemented). The second problem is to find room in the packet header to carry the certificates. Since the number of certificates is variable, one possible solution would be to use the IP option field. It is also possible to allow detected flows to resume normal operation after a "probation period" by extending the certificates to contain the time at which the flows were detected to be misbehaving.

VIII. SIMULATIONS

In this section, we use *ns-2* [16] simulations to evaluate our solution. These simulations aim to demonstrate the ability of our solution to: (1) protect well-behaved flows from misbehaving flows (2) detect flows that consume more resources (bandwidth) than they had reserved, and (3) detect flows that send extra refresh messages.

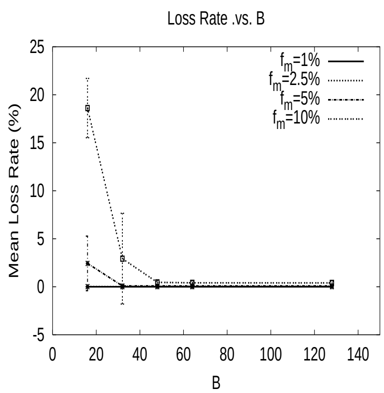

Fig. 9. Loss rate vs B for different f_m for *R-Mon*

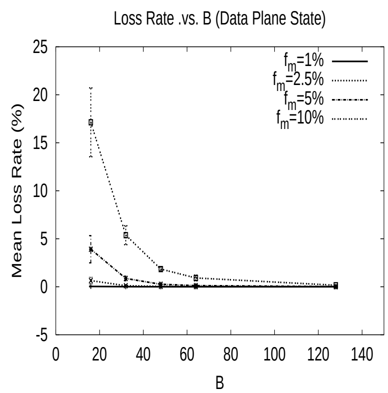

Fig. 10. Loss rate vs B for different f_m for Random Sampling

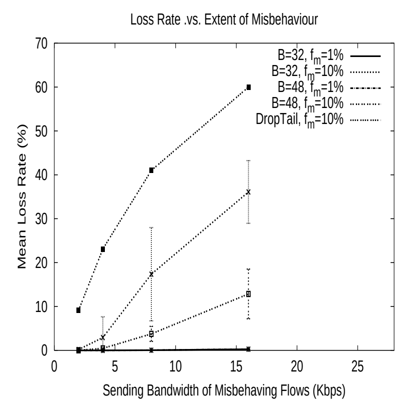

Fig. 11. *R-Mon* Algorithm: Loss rate vs amount of misbehavior for Different B, f_m

In our evaluation we use three metrics: (1) Average loss rate experienced by well-behaved flows, as a measure of the protection offered by our scheme in the presence of misbehaving flows. (2) Fraction of misbehaving flows detected, and (3) The time taken to detect misbehaving flows.

We consider a peak bandwidth allocation service in which all well-behaved flows request and send traffic at a reserved rate of 10 KBps. Each flow is modeled as a CBR source which sends packets with the inter-departure times distributed uniformly between -20% and 20% of the average value. Flows have associated random 32-bit identifiers. In the *R-Mon* scheme, flows are mapped onto A-flows using randomly generated bit-masks for each level. The value of T_{ref} and T_{router} is 5 seconds. Data and control packets are of size 1500 and 500 bytes respectively. For simplicity, we assume that once a misbehaving flow is detected, the flow is barred. Unless otherwise specified, the topology we consider is a single congested link governed by a router at one end, which implements our solution.

A. Detecting Misbehaving Flows on the Data Path

We now evaluate the effectiveness of our detection algorithm to catch flows that misbehave on the data path i.e., flows that exceed their reservation. We consider two cases: (1) a static scenario, where all flows arrive at the same time and are continuously backlogged, and (2) a dynamic scenario, where flows arrive and depart on a continuous basis.

1) Static Scenario: We consider a single congested link traversed by 1024 flows out of which M are misbehaving. We assume that all flows are continuously backlogged, and that all flows start transmitting at the same time. Fig.6 shows the mean time it takes to catch all misbehaving flows for both the random sampling scheme and the *R-Mon* scheme with degree 4. In both cases we assume $B = 48$. The bandwidth of misbehaving flows is 4 times the reserved bandwidth. The number of misbehaving flows M varies from 1 to 128. The graph also plots the theoretical result for the *R-Mon* scheme using the result in Theorem 1. As illustrated by the simulation results, when M is small, the time it takes to catch all misbehaving flows is linear in M. However, as M increases, the probability of an A-flow having more than one misbehaving flow increases and the dependence becomes sub-linear.

In general, the *R-Mon* scheme outperforms random sampling when M is small. This is because random sampling selects the flows to be monitored uniformly no matter whether they are misbehaving or not. Thus, when M is small, random sampling ends up monitoring mostly well-behaved flows. In contrast, *R-Mon* splits only misbehaving A-flows, which results in a more judicious use of the available buffer space.

2) Dynamic Scenario: We next evaluate the ability of our solution to provide protection to well-behaved flows in a dynamic scenario. Flows arrive according to a Poison process with a mean arrival rate of 4 per second. The duration of each flow is drawn from a Pareto distribution with a shape parameter of 1.5. These parameters ensure that the link is being traversed by 1000 flows on average. The duration of each simulation is 6000 seconds of simulated time.

In evaluating the *R-Mon* scheme we vary three parameters: (1) B, the maximum number of flows or A-Flows that can be policed simultaneously, (2) f_m, the fraction of arriving flows that are misbehaving, and (3) bw_m, the ratio of the rate at which misbehaving flows send data to their reserved rates. Thus, a completely reserved link that is traversed by M misbehaving flows will receive extra traffic at a rate of $M * (bw_m - 1) \cdot 10$ KBps. Note that the expected loss rate caused by the misbehaving flows in the absence of our solution would be $(\frac{1}{f_m(bw_m-1)} + 1)^{-1}$. For instance, if 10% of flows misbehave and each of them sends traffic at a rate 4 times the reserved rate (i.e., $bw_m = 4$), the resulting loss rate is 23%! In the following discussion, bw_m is 4 unless otherwise specified.

First, we investigate the relationship between the degree k of the A-tree and B. Fig.8 shows the loss rate of well-behaved flows for $B = 32$ and 48. These results suggest that the optimal choice of the degree k in these cases is 4. This is consistent with the analytical result presented in Theorem 6 (in Section VI), which says that the optimal degree is 4 (see Fig.7) when there are 1000 flows in the system. Note that we use bit-masks to map flows onto A-flows; so the degrees are powers of two. And the nearest such degree to the the optimal point in the theoretical graph is 4.

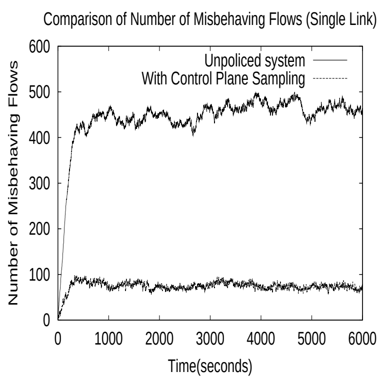

Fig. 12. Number of Misbehaving Flows in a Policed and Un-policed Control Plane

Topology	Avg. Drop Rate	Drop-tail Drop Rate
A-*R-Mon*	1.6%	23%
A-Sampling	1.79%	23%
B-*R-Mon*	0.052%	25.34%
B-Sampling	0.057%	25.34%
C-*R-Mon*	0.53%	24.28%
C-Sampling	0.27%	24.28%
A-*R-Mon*(5% overprovisioning)	0.016%	20%
A-Sampling(5% overprovisioning)	0.024%	20%

Fig. 13. Multiple Link Topology Results

We next investigate the relationship between B and f_m. Fig.9 plots the loss rate versus B for different values of f_m. Note that in this case, $B = 48$ represents the point of diminishing returns; once B exceeds 48, the decrease in the loss rate is minimal. Using the parameters of the simulation with the result presented in Theorem 1, we obtain B to be around 24 which is only half as much as 48. We believe that this is due to the inefficiency of the *R-Mon* scheme that we used (the analytical result gives the performance of an optimal algorithm). For comparison purposes, Fig.10 plots the loss rate experienced by the well-behaved flows for the random sampling scheme. While both algorithms are effective in reducing the loss rates, the *R-Mon* scheme has the edge in most cases.

Finally, we investigate the impact of increasing the rate at which misbehaving flows send data on the loss rate experienced by well-behaved flows. Fig.11 plots the loss rate versus bw_m. When $f_m = 1$ there is little increase in the loss rate as bw_m increases. This is because B is large enough to catch a misbehaving flow almost as soon as it arrives. However, when $f_m = 10\%$, we see a linear increase in the loss rates with bw_m. This is to be expected since the *R-Mon* scheme is not very sensitive to the amounts of data sent by misbehaving flows (more specifically, if all misbehaving flows increase their sending rate by the same factor, no appreciable difference will be seen in the rate at which misbehaving flows are detected). The graph also plots the corresponding loss rates on a drop-tail link with no policing. This shows that our scheme represents a significant improvement over a simple drop-tail queue. We also note that in all the simulations reported here, no well-behaved flows were wrongly classified as misbehaving.

B. Control Plane Misbehavior

We now consider flows that misbehave by sending more refresh messages than specified by the signaling protocol. In particular, well-behaved flows send one refresh message and misbehaving flows send $x_{ref} > 1$ refresh messages every T_{ref} seconds. To stress our solution, we also assume that the misbehaving flows send data at a rate which is x_{ref} times larger than their reservation. This represents the maximum amount of data the misbehaving flows can transmit without being caught on the data plane.

On the data plane, misbehaving flows never exceed their (inflated) reservations. Therefore the loss rate experienced by the well-behaved flows is negligible. The impact of the misbehaving flows is that flows arriving into the system will be denied admission. Thus, to evaluate the efficacy of the detection scheme on the control plane, we use the number of misbehaving flows that are caught, as the main metric. In scenarios where the percentage of misbehaving flows is 10% and at least 48 flows can be monitored, the control plane mechanism is able to detect 99% of the misbehaving flows.

Fig.12 plots the number of misbehaving flows in the system in the presence and absence of policing on the control plane. The percentage of misbehaving flows is 40%. The reason for using such a large percentage is to better illustrate the robustness of the control plane. As shown in Fig.12, the number of misbehaving flows that are not caught is around 10% in steady state. It is important to note that this result does not mean that 10% of the misbehaving flows are never caught; it only means that on an average a misbehaving flow is caught one fourth into its lifetime. In this simulation, we use $B = 8$, $k = 4$ and the mean arrival rate of flows λ is 2.

C. *Multiple Links*

All results presented so far are for a simple network topology involving a single congested link. Now, we show that our solution is also robust and effective in the case of more complex topologies. To illustrate this point, we use the three topologies shown in Fig.14. In all simulations, the overall fraction of misbehaving flows arriving in the system is 10%. The congested links are shown with solid lines. Each flow traverses one congested link in topology A, two congested links in topology B, and four congested links in topology C. Also, to illustrate the incremental deployability of our solution, we assume that only the congested routers implement flow monitoring and containment.

Fig.13 summarizes the results for each of the three topologies. We make several observations. First, even in the absence of over-provisioning, i.e., when a router allocates 100% of its capacity, the loss rate never exceeds 1.8%. The reason why our solution performs better for topologies B and C is that misbehaving flows are caught faster as they traverse multiple routers implementing our solution; in general the time taken to detect a misbehaving flow decreases with the number of routers on its path that implement our solution. Second, adding a 5% overprovisioning reduces the loss rate by almost two orders of magnitude even in the case of Topology A. The over-provisioning factor of 5% is computed as explained Section IV-A.1.

Third, both random sampling and *R-Mon* schemes perform similarly, with *R-Mon* performing marginally better in the case of topologies A and B, and with random sampling performing marginally better in the case of topology C. This is because random sampling reaps full benefit when the number of routers implementing our solution (n) on the flow's path increases. Indeed, with random sampling, the time to catch a flow is roughly inversely proportional to n. In contrast, *R-Mon* is limited by the fact that it has to walk down the *A-Tree* until it detects a misbehaving flow.

Finally, these results illustrate the ability of our solution to be incrementally deployed (as long as the bottleneck links perform admission control). This is because our solution does not require any coordination among the routers in either performing the admission control or in detecting the misbehaving flows.

IX. Conclusions and Future Work

In this paper, we have described a scalable and robust architecture for providing bandwidth allocation in the Internet. Our solution does not require routers to maintain per-flow state on either the data or control plane. The paper's main contribution is to propose two techniques for robustness against malicious users on both the data plane and the control plane. For the control plane we propose a lightweight certificate-based protocol that enables routers to detect users who try to evade admission control, deny service to well-behaved flows, or violate the signaling protocol. For the data plane, we propose a recursive monitoring algorithm that is able to detect misbehaving flows that exceed their reservations, while maintaining minimal state. The simulation results indicate that these mechanisms maintain the quality of service (in this case, bandwidth) of the flows which conform to their reservations, in the presence of a significant, but limited, number of misbehaving flows. The most important area of future work is to extend our solution to accommodate bursty traffic sources and multicast.

X. Acknowledgments

Prof. Randy Katz, L. Subramanian, Bhaskaran Raman, K.L. Karthik and Weidong Cui gave us insightful comments on earlier versions of this report. L. Subramanian, Bhaskaran Raman, Ramakrishna Gummadi and Sharad Agarwal provided us much-needed CPU cycles on their machines. We thank them all.

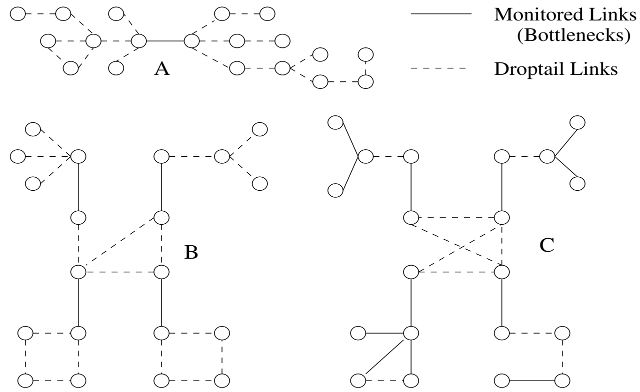

Fig. 14. Complex Topologies

REFERENCES

[1] S. Jamin, P. Danzig, S. Shenker and L. Zhang. A Measurement-based Admission Control Algorithm for Integrated Services Packet Networks. *ACM SIGCOMM'95*, Sep. 1995.

[2] A. Parekh and R. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case. *IEEE/ACM TON*, 1(3):344–357, June 1993.

[3] V. Jacobson et al., An expedited forwarding PHB, June 1999. Internet RFC 2598.

[4] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. *ACM SIGCOMM Internet Measurement Workshop'01*, Nov. 2001.

[5] F. Baker et al., Aggregation of RSVP for IPv4 and IPv6 Reservations, September 2001, Internet RFC 3175.

[6] R. Braden et al., "Integrated Services in the Internet Architecture: An Overview", June 1994, Internet RFC 1633.

[7] D. Clark, "The Design Philosophy of the DARPA Internet Protocols", *ACM SIGCOMM '88*, pp. 106-114, August 1988.

[8] W. Feng et al., "Blue: A New Class of Active Queue Management Algorithms", Technical Report UM CSE–TR–387–99, University of Michigan, 1999.

[9] R. Guerin, S. Blake, S. Herzog, "Aggregating RSVP-Based QoS Requests", Internet Draft, November 1997.

[10] K. Nichols , V. Jacobson, and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", July 1999, Internet RFC 2638.

[11] S. Shenker, C. Patridge and R. Guerin, "Specification of Guaranteed Quality of Service", September 1997, Internet RFC 2212.

[12] I. Stoica and H. Zhang, "Providing Guaranteed Services Without Per Flow Management", *ACM SIGCOMM'99*, Sep 1999.

[13] I. Stoica, S. Shenker and H. Zhang, "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks", *ACM SIGCOMM'98*, Sep. 1998.

[14] I. Stoica, H. Zhang and S. Shenker, "Self-Verifying CSFQ", To appear in *Proceedings of INFOCOM'02*, New York, May 2002.

[15] J. Wroclawski, "Specification of the Controlled-Load Network Element Service", Sep. 1997, Internet RFC 2211.

[16] "Network Simulator 2", http://www.isi.edu/nsnam/ns/