

Controlling High Bandwidth Aggregates in the Network

Ratul Mahajan, Steven M. Bellovin, Sally Floyd,
John Ioannidis, Vern Paxson, and Scott Shenker

AT&T Center for Internet Research at ICSI (ACIRI)
and AT&T Labs Research*

February 5, 2001- DRAFT

Abstract

The current Internet infrastructure has very few built-in protection mechanisms and is therefore quite vulnerable to attacks and failures. In particular, recent events have illustrated the Internet's vulnerability to both Distributed Denial of Service attacks and "flash crowds" in which one or more links in the network (or servers at the edge of the network) become severely congested. In both flash crowds and DDoS attacks the congestion is not due to a single flow, nor to a general increase in traffic, but to a well-defined subset of the traffic – an *aggregate*. This paper discusses mechanisms for detecting and controlling such high bandwidth aggregates. Our approach involves both a local mechanism for detecting and controlling an aggregate at a single router, and a cooperative *pushback* mechanism in which a router can ask adjacent routers to control an aggregate upstream. These mechanisms, while certainly not a panacea, should provide relief from some types of DDoS attacks and flash crowds.

1 Introduction

In the current Internet, when a link is persistently overloaded all flows traversing that link experience significantly degraded service over an extended period of time. Protection mechanisms that could minimize the effects of such congestion would greatly increase the reliability of the Internet infrastructure. Persistent overloads can arise for several reasons, and each requires a different form of protection.

First, persistent overloads can result from a single flow not using end-to-end congestion control and continuing to transmit despite encountering a high packet drop rate. There is a substantial literature on mechanisms to cope with such *ill-*

*Ratul Mahajan is from University of Washington (work done while at ACIRI); Steven M. Bellovin and John Ioannidis are from AT&T Labs Research; Sally Floyd, Vern Paxson and Scott Shenker are from ACIRI. The email addresses are ratul@cs.washington.edu, smb@research.att.com, floyd@aciri.org, ji@research.att.com, vern@aciri.org, shenker@aciri.org

This is a draft, and should not be mirrored, archived, or redistributed.

behaved flows (where, by *flow*, we mean a stream of packets sharing IP source and destination addresses, protocol field, and source and destination port numbers). Routers with per-flow scheduling, such as Fair Queueing [DKS89] and related algorithms [SV95], isolate flows from each other and thereby prevent a single flow from degrading the service experienced by other flows. Similar results can be achieved by more recently developed approaches that use preferential dropping to control high bandwidth flows [SSZ98, PPP00, MF00].

As was seen on the transatlantic links a few years ago, persistent overloads can also be due to a general excess of traffic [ILS99]. While better active queue management techniques [FJ93] may be of some use, there is little one can do to protect inadequately provisioned links. However, even when all links are adequately provisioned, and all legitimate flows are using conformant end-to-end congestion control (or, equivalently, all routers have mechanisms to protect against ill-behaved flows), persistent congestion can still occur. Two examples of this are *denial of service* attacks (DoS) and *flash crowds*.

DoS attacks occur when a large amount of traffic from one or more hosts is directed at some resource of the network (*e.g.*, a link or a web server). This artificially high load denies or severely degrades service to legitimate users of that resource. The current Internet infrastructure has few protection mechanisms to deal with such DoS attacks, and is particularly vulnerable to distributed denial of service attacks (DDoS), in which the attacking traffic comes from a large number of disparate sites. A series of DDoS attacks occurred in February 2000 [Gar00] to considerable media attention.

Flash crowds occur when a large number of users try to access the same server simultaneously, overwhelming the available resources. In addition to the overload at the server itself, the traffic from such flash crowds can overload the network links and thereby interfere with other, unrelated users on the Internet. For example, degraded Internet performance was experienced during a Victoria's Secrets webcast and during the NASA Pathfinder mission.

While the intent and the triggering mechanisms are quite different for DoS attacks and flash crowds, from the network's perspective these two cases are quite similar. The persistent congestion is not due to a single well-defined flow, nor is it

due to an *undifferentiated* overall increase in traffic. Instead, there is a particular set of packets causing the overload, and these offending packets – which we will call an *aggregate* – are spread across many flows. The resulting *aggregate-based congestion* cannot be controlled by conventional per-flow protection mechanisms. In this paper we propose control mechanisms that work on the granularity of aggregates. These “Aggregate-based Congestion Control” (ACC) mechanisms fall between the traditional granularities of per-flow control (which looks at individual flows) and active queue management (which does not differentiate between incoming packets).

More specifically, we define an *aggregate* as a collection of packets from one or more flows which have some property in common. This property could be anything from destination or source address prefixes to a certain application type (streaming video, for instance). Other examples of aggregates are TCP SYN packets and ICMP ECHO packets. An aggregate could be defined by a property which is very broad, such as TCP traffic, or very narrow, such as HTTP traffic going to a specific destination.

To reduce the impact of congestion caused by such aggregates, we propose two related ACC mechanisms. The first, “local” aggregate-based congestion control, consists of an *identification* algorithm used to identify the aggregate (or aggregates) causing the congestion, and a *control* algorithm that then reduces the traffic sent by this aggregate to a reasonable level. As we will discuss, there are many situations in which local aggregate-based congestion control would, by itself, be quite effective in preventing aggregates from significantly degrading the service delivered to other traffic.

In some cases, however, it may be beneficial to control the aggregate closer to its source(s). The second ACC mechanism, “pushback”, allows a router to request adjacent upstream routers to rate-limit traffic corresponding to the specified aggregates. This mechanism is particularly effective if the aggregate is not generated by sources spread across the whole network, as a result of which upstream links of routers contribute unevenly to the aggregate.

These ACC mechanisms are intended to protect the network from persistent and severe congestion due to rapid increases in traffic from one or more aggregates. We envision that these mechanisms would be invoked rarely, and we emphasize that these mechanisms are not substitutes for adequately provisioning links or for end-to-end congestion control. Nonetheless, we believe that introducing control mechanisms at this new level of granularity – aggregates – may provide important protection against flash crowds, DoS attacks, and other forms of aggregate-based congestion.

The organization of this paper is as follows. Section 2 gives an overview of ACC and pushback. In Section 3 we describe some related work done to tackle the problem of DoS attacks

and flash crowds. Section 4 describes ACC in more detail. We discuss the pushback mechanisms in detail in Section 5, followed by some simulation results in Section 6. Section 7 evaluates the advantages and disadvantages of pushback, and discusses of several open issues related to ACC.

2 Overview of ACC

In this section we give an overview of our two proposed ACC mechanisms: *Local ACC*, in which a router deals with sustained overload by itself, and *Pushback*, an extension to Local ACC in which a router signals other routers upstream to control a particular aggregate on its behalf. The mechanisms are then explored in detail in Section 4 and Section 5.

We can think about an ACC mechanism running in a router as consisting of the following sequence of decisions:

1. Am I seriously congested?
2. *If so*, can I identify an aggregate responsible for an appreciable portion of the congestion?
3. *If so*, to what degree do I limit the aggregate? Do I also ask upstream routers to limit the aggregate?
4. *And if I decide to deal with it*, when do I stop?

Each of these questions requires an algorithm for making the decision. Each is also a natural point to inject *policy* considerations into the decision making. The space of possible policies (e.g., who to treat better than whom, whom to trust, what applications should get at most how much bandwidth, how to perhaps incorporate past history) is very large, and we do not attempt to explore it in this paper. Instead, we assume simple policies in order to focus on developing and understanding the mechanisms.

To answer the question “am I seriously congested?” our proposed mechanism periodically monitors each queue’s packet drop rate to see if it exceeds a (policy-specific) threshold. The monitoring interval is jittered, both to avoid synchronization effects [FJ94] and to resist an attacker intent on predicting the response patterns of ACC in the presence of a DoS attack.

When serious congestion is detected, the router attempts to identify the aggregate, or aggregates, responsible for the congestion. Identifying the offending aggregate(s) is a tricky problem to solve in a general fashion, for three reasons. First, the overload may be chronic, due to an under-engineered network, or unavoidable, e.g. as a shift in load caused by routing around a fiber cut. These lead to *undifferentiated congestion* not dominated by any particular aggregate. Second, there are many possible dimensions in which traffic might cluster to form aggregates: by source or destination address (e.g., a flash crowd attempting to access a particular server, or its

replies back to them), address prefix (a flooding attack targeting a site or a particular network link), or a specific application type (a virulent worm that propagates by email, inadvertently overwhelming other traffic). Third, if the congestion is due to a DoS attack, the attacker may vary their traffic as much as possible to complicate the router’s detection of high-bandwidth aggregates.

We propose that routers identify aggregates by applying clustering to a sample of their high volume traffic, which they can attain by sampling drops from a randomized discard mechanism such as RED [FJ93]. We discuss the specifics of a possible clustering algorithm in Section 4.1. Note that if the clustering algorithm fails to find a narrowly defined aggregate, we conclude that the congestion is undifferentiated and take no action.

Analogous to *attack signature* for describing various forms of malicious activities, we use the term *congestion signature* to denote the aggregate(s) identified as causing congestion. It is important to note that when constructing congestion signatures, the router does not need to make any assumptions about the malicious or benign nature of the underlying aggregate (which may not in fact be possible in the face of a determined attacker). If the congestion signature is too broad, such that it encompasses additional traffic beyond that in the true high-bandwidth aggregate, then we refer to the signature as incurring *collateral damage*. In this case, restricting the bandwidth of the identified aggregate can increase the already high packet drop rate seen by the legitimate traffic within the aggregate, while easing the burden on the legitimate traffic that did not fall within the aggregate. Narrowing the congestion signature, and thus minimizing collateral damage, is one of the goals of our approach.

We now turn to the question of to what degree the router should limit an aggregate’s rate, and the mechanism by which it does so. We argue that there is no useful, policy-free equivalent to max-min fairness when applied to aggregates; no one would recommend for best-effort traffic that we give each destination prefix or application type an equal share of the bandwidth in a time of high congestion. Instead, the goal is to rate-limit the identified aggregate sufficiently to protect the other traffic on the link from the congestion caused by the aggregate. Here, “sufficiently” is chosen such that, for all the aggregates we are currently rate-limiting, we restrict them so that their total arrival rate plus that of other traffic arriving at the queue maintains an ambient drop rate in the output queue of at most the configured target value (Section 4.2).

Figure 1 shows the proposed rate-limiting architecture. The rate-limiter acts as a filter at the entry to the regular FIFO output queue. When a packet arriving at the output queue is identified as a member of the aggregate, it is passed to the rate-limiter, which decides whether to drop the packet or add the packet to the output queue. Once past the rate-limiter, the packet loses any identity as a member of the aggregate and

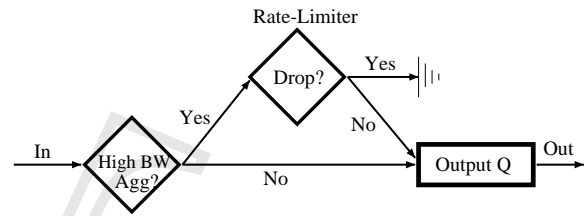


Figure 1: **The rate-limiting architecture.** Different rate-limiters make the drop decision differently.

is treated the same as any other packet arriving at the output queue. Because packets that pass the rate-limiter are treated as regular arrivals to the output queue, rate-limiting cannot result in preferential treatment for the packets in the aggregate. The rate-limited aggregates can get preferential treatment when they are allocated a fixed bandwidth share irrespective of the general congestion levels at the output queue. Packets belonging to the aggregate can be dropped either by the rate-limiter, or by the output queue itself if the queue is full or the router employs an active queue management technique such as RED [FJ93].

We next turn to the possibility of using pushback to control an aggregate. If a router can identify a sufficiently narrow congestion signature (with minimal collateral damage), and if the aggregate responds to packet drops as a congestion signal (flash crowds should have this property), then using pushback buys little over using purely local ACC. This is because if the aggregate is squeezing out other traffic upstream, whether to the same destination or cross traffic, then the corresponding upstream router would also invoke ACC to control the aggregate; so we don’t need to invoke pushback to protect traffic upstream.

But if either of the above conditions do not hold, then using pushback can realize considerable advantages. In addition, the decision as to when to use pushback will likely have a large policy component, and we do not address in this work how a router might make the decision, other than to note that it might be based on observing an exceptionally high drop rate, or the congestion signature matching one configured into the policy as indicating a likely DoS attack. Pushback is intended to work either with or without human intervention, and potentially across administrative boundaries. Pushback can also be initiated by an overloaded server so that, in cases where a DoS attack was not causing congestion but was overloading a server, the benefits of pushback would still be available.

Pushback works by the congested router requesting its adjacent upstream routers to rate-limit traffic corresponding to a given aggregate. This *pushback message* is only sent to immediate upstream routers that send the bulk of the traf-

fic for that aggregate.¹ Throttling the high-bandwidth aggregate closer to the source prevents bandwidth being wasted on packets that are destined to be dropped later on in the network anyway. In addition, by concentrating the rate limiting on the upstream links that carry the bulk of the traffic within the aggregate, pushback can restrict the degree to which a DoS attack denies service to legitimate traffic, since legitimate traffic on the other links will not suffer rate-limiting.

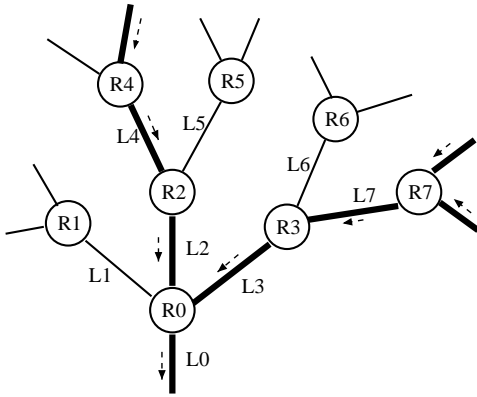


Figure 2: **Illustration of pushback.**

For example, consider the partial network topology shown in Figure 2. The paths used by most of the traffic in the high-bandwidth aggregate are shown in bold, and the direction is as indicated by the arrows. The destination of the aggregate is some host D (not shown) which is reached using $L0$. Thus, most of the traffic covered by the attack signature comes from links $L2$ and $L3$, with very little coming from link $L1$.

Assume that the link $L0$ in Figure 2 is highly congested, and as a result $R0$ identifies the high bandwidth aggregate. By using local ACC, $R0$ can protect the traffic not going to D . But with local ACC only, traffic going from $L1$ to D is not protected; pushback is needed to protect that traffic. Pushback in this case will propagate from $R0$ to $R2$ and $R3$. Subsequently, pushback will propagate upstream to $R4$ and $R7$. Pushback will not be invoked from $R0$ to $R1$. (Note that the path taken by pushback is the reverse of that taken by the high-bandwidth aggregate, and so pushback incidentally provides a form of *traceback* if the source addresses in the aggregate are spoofed [FS00].) Pushback of rate limiting to upstream routers $R2$ and $R3$ helps protect the traffic to D which comes in from $L1$. Similarly, pushing back further up to $R4$ from $R2$ and to $R7$ from $R3$ saves traffic coming along links $L5$ and $L6$ respectively.

¹Clearly, pushback messages require authentication, lest they provide a powerful denial-of-service mechanism themselves! One cheap form of authentication would be to require the messages to arrive with a TTL of 255, that is, restrict their propagation to a single IP hop, in which case forging a pushback message requires compromising either a router or the network's physical security, either of which already gives an attacker major opportunities to do damage.

If source addresses could be trusted, then in some cases the congested router could narrow the attack signature itself, without pushback, by identifying both the source and the destination address prefixes responsible for the bulk of the traffic in the identified aggregate. In this case, the traffic in the more narrowly-defined aggregate could be dropped at the congested router itself, without invoking pushback. However, pushback would still be useful to reduce upstream bandwidth wasted on traffic that will only be dropped downstream.

In addition, if the offending traffic within an aggregate is heavily represented on some upstream link in the network, but the congested router cannot identify this subset of the aggregate on the basis of source IP address prefixes alone, then pushback is necessary to narrow the attack signature, *even if source addresses are genuine*.

The last question posed at the beginning of this section was: “when do I stop?” For local ACC, the answer is simple: as discussed in Section 4.4, the router continues to monitor whether it is experiencing significant congestion, and, if so, determines the major aggregate(s) responsible. If the router is no longer significantly congested, or if a particular aggregate being limiting is no longer one of the main responsible aggregates, then the router stops limiting the aggregate. (Clearly, we need to worry about an attacker predicting this decision in order to evade ACC.)

For pushback, however, the decision becomes more difficult, because the router must distinguish between no longer seeing much traffic from the aggregate because it is being limited upstream, versus because the aggregate has stopped sending much traffic. Disambiguating these two cases in turn motivates the need for *feedback* messages that the upstream routers send out reporting on how much traffic from an aggregate they are still seeing; see Section 5.4.

3 Related Work

In this section we discuss the various existing techniques to deal with flash crowds and DoS attacks. Some of the techniques for dealing with DoS attacks focus on protecting the network by dropping malicious packets; other techniques try to solve the *traceback* problem of tracing the attack back to the source(s). The traceback problem arises because the source IP addresses in IP packets are easily spoofed in the current Internet. When the source addresses are spoofed, a successful traceback would let the victim (and the network) find the immediate source of the attack. Tracing back the attack to its source allows steps to be taken which stop the attack, and is the first step towards the necessary legal actions to discourage such attacks in the future.

Identifying the machines sending attack traffic does not necessarily lead to finding the ultimate originators of an attack.

But it does allow the network to drop the attack packets near their source, before they damage the rest of the network. In addition, note that identifying these machines is not a *requirement* for preventing the damage caused by an attack; all that's needed is to sufficiently localize them in the topology. For example, if congestion from an attack begins downstream from the immediate sources of the attack, but upstream from the intended victim, then the attack could be detected at the congested router. While it might be difficult for a router to distinguish between malicious and non-malicious traffic, a router can prevent some of the malicious traffic from flooding the victim, and to protect at least some of the non-malicious traffic from the congestion caused by the malicious traffic. Also, in the presence of ACC mechanisms, we expect the damage control (by preferential dropping of the high-bandwidth aggregate) to trigger in much sooner than the time it takes to identify and stop the malicious sources.²

3.1 Identifying the source of an attack

One approach to the traceback problem is to reduce or eliminate the ability to spoof IP source addresses by some form of source filtering. In *ingress filtering* [FS00], an ISP filters out packets with illegitimate source addresses, based on the ingress link by which the packets enter the network. In contrast, *egress filtering* [SAN00] occurs at the exit point of a customer domain, where a router checks whether the source addresses of packets actually belong to the customer's domain. Packets with invalid source addresses are dropped.

While source filtering is increasingly supported as a necessary step in the protection against DoS attacks [ICS00], source filtering is not likely to completely eliminate the ability to spoof source IP addresses. For instance, if source filtering is done at the customer-ISP level, a single machine within the customer network can still disguise itself as any of the hundreds or thousands of machines in the customer domain. Even effective source filtering does not prevent attacks from compromised machines with valid source addresses.

In contrast to source-based filtering, which tries to curb attacks at the source, *traceback* assumes that source addresses can be spoofed, and tries to identify the source(s) of malicious traffic using the network itself. Recent proposals for traceback include a variety packet-marking schemes, i.e., Savage et al., [SWKA00], Song and Perrig [SP01], and Dean et al. [DFS01], as well as Bellovin's ICMP Traceback [Bel00]. In packet-marking the routers use the IPv4 ID field to report information about the edges of the network that the pack-

ets traversed. The collective edge information can then be analysed at the victim to compute the path of an attack. In ICMP Traceback routers, with a very low probability (like 1/20,000), sample packets going through them and send an ICMP message to the destination of the packet. The ICMP message contains the identity of the router itself, contents of the packet and information about adjacent routers. During the times of an attack, these messages will help the victim calculate the path used by the attack traffic.

In another possible traceback scheme suggested in [Sto00], all edge routers would log packets. The data obtained from all the routers is then analyzed to find the ingress point of the attack traffic. Another traceback proposal [BC00] assumes that the victim has an approximate map of the Internet. The victim then asks some selected hosts to flood each incoming link of the router closest to it. By observing the changes in attack traffic, the victim can determine the link used by the attack traffic. This process is then recursively repeated upstream to get closer to the source of the attack.

In the absence of effective source filtering, some form of traceback would be required to identify the ultimate source of an attack. Weaknesses shared by all of the traceback proposals are that the damage done by the attack is not being controlled while the traceback is in progress, and the effectiveness of traceback schemes is reduced as an attack becomes more distributed. We see ACC and Pushback as complementary to both source filtering and to traceback.

Schnackenberg et al. [SDS00] suggest active control of infrastructure elements. Thus, a firewall or IDS that detected some sort of attack could request that upstream network elements block the traffic. There are obvious problems authenticating such requests in the inter-domain case, though work in the field is ongoing.

3.2 Identifying the nature of the attack

Some sites filter or rate-limit all traffic belonging to a certain category to evade particular kinds of attack. An example would be filtering ICMP ECHO messages to prevent the well-known smurf [CER98] attack. Such content-based filtering based on fixed filters can be of use, particularly in the short term, but is by definition limited to the fixed filters already defined. ACC and Pushback are based on using filters which are both dynamic and wider in range.

Input debugging uses attack signatures to filter out traffic at the routers. The victim identifies an attack signature and communicates it to its upstream ISP. The ISP installs a filter on its egress router to the victim, thus stopping the attack traffic. At the same time the ISP identifies the router's incoming interface of the attack, and recursively repeats the process upstream. Determining and controlling the attack traffic all the way to the sources requires cooperation between all

²The fact that ACC, in both its local and pushback incarnations, gently restrains aggregates to the point where they are no longer causing congestion allows ACC to respond rather quickly because the downside of an inaccurate assessment of the offending aggregate is slight. DoS countermeasures that completely shut down the attacking traffic must be much more confident in their identification before they take action.

entities controlling the routers on the paths from sources to victim. This is easier said than done, since the paths often cross administrative boundaries. The solution works on human timescales and is labor intensive. It requires the presence of skilled operators to successfully carry it out (though some ISPs have tools to do some of this work semi-automatically in their networks [Art97]).

Our proposal for Pushback is closely related to input debugging, except that instead of starting from an attack signature from a downstream victim, we would also start with a congestion signature from the congested router itself.

Instead of hop-by-hop input debugging, [Sto00] proposes building an overlay consisting of all edge routers and one or more tracking routers. In case of attacks the input debugging procedure would be carried out along the overlay tunnels. The scheme requires an overlay connecting all the edge routers of an ISP, with appropriate authentication between routers, and changes to global routing tables. Each ISP would use its own overlay system to find the entry and exit points of the traffic in its domain, using human intervention when crossing ISP boundaries.

3.3 Related Work on ACC and Network Congestion

In this section we discuss briefly related bodies of work on web-caching and content distribution infrastructures, scheduling mechanisms, and Quality of Service, and their relationship to ACC.

Web-caching infrastructures and Content Distribution Networks (CDNs) [Dav00] like Akamai[Aka] and Digital Island [Dig] are powerful mechanisms for preventing flash crowds from congesting the network. IP Multicast and application-level multicast are additional tools for accommodating flash crowds without creating congestion in the network, for a different set of applications. However even the combination of multicast, caching infrastructures, and CDNs may not be sufficient to completely prevent network congestion from flash crowds. For example, flash crowds could occur for traffic not carried by the prevailing CDNs, or for traffic marked as uncacheable by the origin server, or for traffic that is not suitable for multicast distribution. Internet slowdowns could still be caused by an event or site which witnesses an unprecedented “success” for which neither it nor the related infrastructure is prepared.

There is a considerable body of work on scheduling and preferential dropping mechanisms that have some relationship to ACC. Per-flow scheduling mechanisms include Fair Queuing [DKS89] and Deficit Round Robin [SV95]. There is a growing body of work on using drop preference to approximate per-flow scheduling [SSZ98, PPP00] or to protect conformant flows from flows that do not use end-to-end conges-

tion control [FF97, LM97]. However, flow-based congestion control and scheduling mechanisms are not solutions for aggregate-based congestion control, since an aggregate could be composed of many flows which are conformant individually. CBQ [FJ95] is a class-based scheduling mechanism in which aggregates can be limited to a certain fraction of the link bandwidth in a time of congestion. However, CBQ is discussed largely for fixed definitions of aggregates, and does not include mechanisms for detecting particular high-bandwidth aggregates in times of congestion.

There is also a substantial body of work on QoS mechanisms like Integrated Services [CSZ92] and Differentiated Services [BBC⁺98] to protect a designated body of traffic from congestion caused by lower-priority or best-effort traffic. Such QoS mechanisms could be a critical component in protecting designated traffic from congestion caused by best-effort flash crowds or DoS attacks.

4 Aggregate-Based Congestion Control

This section describes the architecture and the algorithms used by the router to detect and control high-bandwidth aggregates in times of sustained high congestion. A more formal description (pseudocode) of the algorithms can be found in Appendix B.

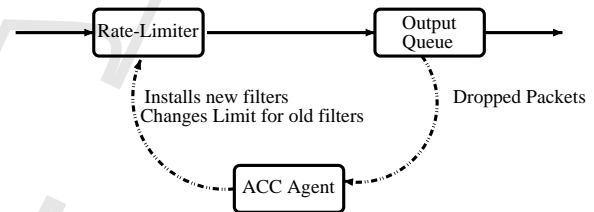


Figure 3: **Router architecture with ACC Agent.** The solid lines show the packet forwarding path.

The ACC can be broken down into detection and control. In Figure 3, the ACC Agent is responsible for the identification of aggregates and computing a rate limit for them. The actual rate-limiting (by dropping packets) is done by the *Rate-Limiter*. The ACC Agent is not in the fast path used for packet forwarding. On the other hand, packets arrive to the *Rate-Limiter* which determines if it belongs to a rate-limited aggregate. Packets belonging to a rate-limited aggregate may be dropped by the *Rate-Limiter* depending on the arrival rate of that aggregate and the rate limit imposed on it. Packets that survive are forwarded to the output queue. Dropping also takes place at the output queue because of normal congestion. Relevant information (headers) about packets dropped at the output queue is fed into the ACC Agent which uses these

packet drops for identifying high-bandwidth aggregates. Alternately, the output queue can pass random samples to the ACC Agent which can be used in the identification process instead.

The identification process in the ACC Agent is triggered when the output queue experiences sustained high congestion. We define sustained congestion as a drop rate of more than p_{high} over a period of K seconds. During times of sustained congestion, using the packet drop history (or random samples) of the last K seconds, the ACC Agent tries to identify a small number of aggregates responsible for the high congestion. If some aggregates are found, the ACC Agent computes the limit to which these aggregates should be restricted. The limit is computed such that the *ambient* drop rate, that is the drop rate at the output queue (does not take into account the drops in the Rate-Limiter), is brought down to below p_{target} . At the same time this limit cannot be less than the highest arrival rate estimate among aggregates which are not being rate-limited. The ACC Agent then installs the necessary filters at the Rate-Limiter to rate-limit the identified aggregates. The ACC Agent is also responsible for modifying the limit imposed on various rate-limited aggregates based on changes in demand from background traffic.

The following subsections discuss the algorithms for identifying aggregates to be rate-limited, determining the rate, and implementing the rate-limiting. Later sections show how the Pushback of rate-limiting to upstream nodes could be combined with local aggregate-based congestion detection and control to help make finer distinctions between the legitimate and the malicious traffic within an aggregate.

4.1 Identification of High Bandwidth Aggregates

In principle, aggregates can be characterized without a source or destination component; all DNS packets, for instance. However, almost all DoS attacks and flash crowds have either a common source or common destination (prefix). At places in the network where source addresses cannot be trusted, it is a dangerous proposition to filter based on them.

In this paper we present a technique to identify high-bandwidth aggregates based on the destination address. This is only one of many possible algorithms for identifying high-bandwidth aggregates; more accurate and flexible algorithms are a subject of further research.

Almost all big web sites use from one to twenty IP addresses close to each other. If one were to specify a prefix which characterized all the IP addresses in use, this prefix would be longer than 24 bits in most cases. Even the sites which need less than 24 bits in their prefix envelopes can be better characterized by multiple 24+ bit envelopes. With this in mind we designed the identification technique.

Based on the drop history (or random samples) draw out a list of high-bandwidth addresses (32-bit); for example, addresses with more than twice the mean number of drops. Now cluster these addresses into 24-bit prefixes. For each of these clusters try obtaining a longer prefix that still contains most of the drops. This can be easily done by walking down the prefix tree having this 24-bit prefix at the root. At each step a heavily biased branch gives a longer prefix with most of the weight. We also try to merge prefixes that are closely related to each other. For example, two adjacent 24-bit prefixes can be described by a single 23-bit prefix. Multiple clusters can still be formed for sites which have spaced-out IP addresses, but this should be fine since using longer prefixes describes the congestion signature better and would not punish a larger category of traffic. All these clusters are then sorted in decreasing order based on the number of drops associated with them. The number of drops also gives us an arrival rate estimate for each cluster. The algorithm to decide how many clusters should be rate-limited in order to decrease the ambient drop rate to below p_{target} is described in the next section.

Given the fact that access links have much less capacity than the backbone links, they are more likely to be congested during times of DoS attacks and flash crowds. The identification of high-bandwidth aggregates is easier in such cases. For instance, the aggregates for the congested router can simply be the prefixes present in its routing table.

The use of ACC should be restricted to special times, such as DoS attacks and flash crowds, when the network needs to protect the rest of the traffic from a few aggregates. Unlike those for flows, the policy level objectives of fairness among aggregates are not clear. Ideally, the ACC mechanisms should not rate-limit any aggregate during times of undifferentiated congestion caused by underprovisioned links or hardware failures. In the absence of effective methods for distinguishing between aggregate-based and undifferentiated congestion, we need to place an upper bound on the number of aggregates that are rate-limited simultaneously. Once we have greater understanding of the traffic composition and behavior during DoS attacks and flash crowds we can tune the ACC mechanism such that it does not identify any aggregate in times of undifferentiated congestion.

4.2 Determining the Rate Limit for Aggregates

Using the list of high-bandwidth aggregates obtained above, the ACC Agent decides how many aggregates to rate-limit and computes the limit to be imposed on them. (This calculation is given in Figure 17 of Appendix B.) The ACC Agent calculates R_{excess} , the excess arrival rate at the output queue, that is, the amount of traffic that would have to be dropped before the output queue (at the Rate-Limiter) to bring the ambient drop rate down to p_{target} . The goal is to rate-limit i top aggregates and preferentially drop from them an amount

of traffic equal to R_{excess} . When i aggregates are chosen, the limit L is computed such that $\sum_{k=1}^i (arrRate_k - L) = R_{excess}$, where $arrRate_k$ is the arrival rate estimate of aggregate k . L should be less than the arrival rate estimate of $i + 1$ -th aggregate.

In the presence of policy constraints, the above computation would have to be modified slightly. For instance, the router could be configured not to give less than B Mbps to a certain aggregate in times of congestion. Such policy level decisions have to be honored in the rate limit calculation (an easy extension of the above algorithm can do this).

4.3 Rate-limiter

The rate-limiter is a crucial component in ACC-enabled routers. It is responsible for classifying packets, rate-limiting the ones belonging to a rate-limited aggregate, and measuring the arrival rate of the rate-limited aggregates. This section discusses the properties of the rate-limiting architecture shown in 1 and describes a mechanism for implementing the rate-limiter.

The rate-limiter needs to be light-weight and efficient to implement as it sits in the forwarding fast path. Figure 1 shows the rate-limiting architecture. Because the rate-limiter is a pre-filter before the output queue that merely decides whether or not to drop each arriving packet in the aggregate, it is consistent with FIFO scheduling in the output queue. Unlike strict lower priority queues, it will not starve the identified aggregates. Packets from the rate-limited aggregate that pass the pre-filter are put in the output queue itself. This means that rate-limited aggregates are not given preferential treatment, or protected from the normal congestion occurring in the output queue. To ensure that the rate-limited aggregates are protected from each other, the drop decision for each aggregate is taken independently based on the state for that aggregate.

In the next section we discuss the virtual queue, the mechanism that we use for rate-limiting. Appendix A describes preferential dropping, an alternate mechanism for the rate-limiter, and contrasts a preferential dropping mechanism with a virtual queue. Preferential dropping and virtual queues differ in the procedure for making a rate-limiting (or, in other words, drop) decision. However, both mechanisms only drop packets from the aggregate when the aggregate's arrival rate to the rate-limiter is above the specified limit. For our simulations, we use a virtual queue, implemented as a simple token bucket, for the rate-limiting mechanism.

4.3.1 Virtual Queue

One possible implementation of a rate limiter is a virtual output queue. An alternative, preferential dropping, is discussed

in Appendix A.

A virtual queue can be thought of as simulating a queue, without actually queuing any packets. The service rate of the simulated queue is set to the specified bandwidth limit for the aggregate, and the queue size is set to the tolerated burst size.

When a packet arrives at the rate-limiter, the rate-limiting mechanism simulates that packet arriving at the virtual queue. Packets that would have been dropped at the virtual queue are dropped by the rate-limiter. Packets that would have been queued at the virtual queue are not dropped by the rate-limiter, but are forwarded to the real output queue.

A virtual queue can simulate either a simple tail drop queue or a queue with active queue management. A virtual queue that simulates tail drop behavior can be thought of as a token bucket, with the fill rate of the token bucket set to the bandwidth limit of the rate-limiter, and the bucket size of the token bucket set to the tolerated burst size for the rate-limiter.

When a packet arrives to the rate-limiter, the rate-limiter checks if there are matching tokens in the token bucket. If so, then a matching set of tokens is removed from the token bucket, and the packet passes the rate-limiter. If there are insufficient tokens in the token bucket, then the arriving packet is dropped by the rate-limiter.

4.3.2 Narrowing the Congestion Signature

In the discussion above, the aggregates identified by the ACC Agent are purely destination-based. In fact, the rate-limiter can do more sophisticated narrowing that, in times of specialized attacks, can result in dropping more of the attack traffic within the aggregate. An example would help to clarify this point. Within an aggregate being rate-limited, the rate-limiter could be configured not to allow more than 20% of traffic within that aggregate go to ICMP ECHO packets. That is, based on drops from the aggregate, the rate-limiter can decide whether the fraction of ICMP ECHO packets within the aggregate is more than 20%. If that is indeed the case, the rate-limiter can invoke a more specialized form of rate-limiting in which it would not let more than 20% of the aggregate's traffic go to ICMP ECHO packets. The earlier constraint of not allowing more than L Mbps of aggregate through should still hold lest the attacker try to fool the router by continuously changing the attack signature. This narrowing of the rate-limiting can be easily achieved by placing another virtual queue in front of the aggregate's virtual queue with a service rate of $0.2 * L$ Mbps for ICMP ECHO packets.

Specialized rate-limiting as described above can be very useful in cases of attacks like the SYN attack [CER96] or the smurf attack [CER98]. It is more dynamic than fixed filters that never let ICMP ECHO packets pass through or impose a low rate-limit on them at all times. Depending on the capabilities of the router, narrowing can be extended to include

a wide variety of signatures based on even application-level headers.

4.4 Reviewing the Rate-Limit

Periodically, the router reviews the rate-limit imposed on the aggregates. Based on the new estimate of the arrival rate into the output queue, a new limit is calculated for the rate-limited aggregates. As mentioned before, this limit should not be below the highest arrival rate among aggregates that are not rate-limited. Aggregates that have had an arrival rate less than the limit for some number of refresh intervals are no longer rate-limited. Similarly, if congestion persists, more aggregates may be added to the list of rate-limited aggregates. It should also be noted that there is no harm done even if rate-limiting continues for some time after the DoS attack (or flash crowd) has subsided because the rate-limiter only drops packets when the arrival rate is more than the specified limit. When the DoS attack subsides, the arrival rate of the aggregate is likely to be much below this limit.

4.5 Simulations

We use a simple simulation to illustrate the behavior of the ACC Agent. Figure 4 shows a simple simulation without ACC, with five aggregates, each composed of multiple CBR flows, with the sending rate of the fifth aggregate varying over time.³ Because this simulation is of CBR flows, rather than of flows using end-to-end congestion control, it is highly unrealistic, and has very simple dynamics; it is included only to illustrate the underlying functionality of ACC.

The two graphs in Figure 4 show that, without ACC, the high-bandwidth aggregate is able to capture most of the link bandwidth. The bottom graph of Figure 4 shows the ambient packet drop rate in the output queue. At time 13 the the sending rate of the fifth aggregate gradually increases, increasing the drop rate and decreasing the bandwidth received by the other four aggregates.

Figure 5 shows the same simulation repeated with ACC enabled. When the ambient drop rate exceeds the configured value of 10%, the ACC Agent attempts to identify an aggregate or aggregates responsible for the high congestion. Within a few seconds the ACC Agent identifies the fifth aggregate, and rate-limits that aggregate sufficiently to control the drop rate in the output queue. The bottom graph of Figure 5 shows the ambient drop rate in the output queue, but does not show the drop rate in the rate-limiter for the fifth aggregate.

³These simulations can be run with the commands “./test-all-pushback slowgrow” and “./test-all-pushback slowgrow-acc” in the tcl/test directory in the NS simulator.

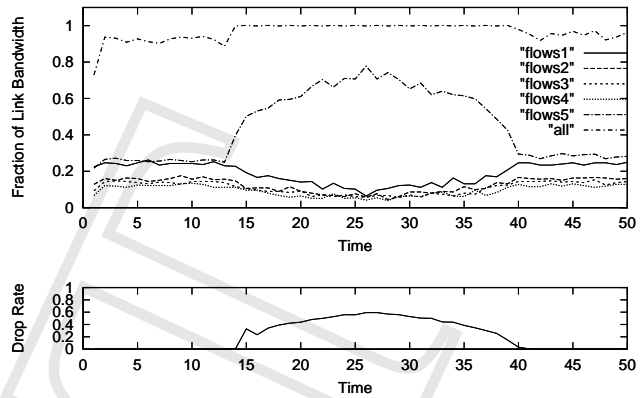


Figure 4: A simulation without ACC.

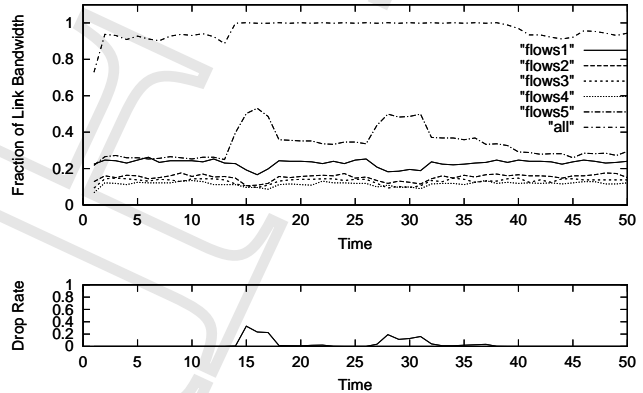


Figure 5: The same simulation with ACC.

5 The Pushback Mechanism

Section 4 described local ACC without pushback; let us now discuss the pushback mechanism in detail. Pushback for an aggregate can be visualized as a tree (or, with multipath routing, possibly a graph), where the congested router initiating the pushback is the root, and the upstream routers rate-limiting the aggregate are the interior nodes and the leaves of the tree. For example, in Figure 2, node R0 is the root of the pushback tree, and nodes R4 and R7 are the leaves.

5.1 Deciding when to Invoke Pushback

After detecting aggregate-based congestion, the ACC Agent must decide whether to invoke Pushback by calling the Pushback Agent at the router. The ACC Agent has information only about its own output queue, while the Pushback Agent coordinates information from diverse input and output queues, and sends and receives pushback messages from neighboring routers.

Two situations warrant the invocation of pushback. The first is when the drop rate for an aggregate in the Rate-Limiter re-

mains high for several seconds (because the arrival rate for the aggregate is much higher than the limit imposed on it).⁴ The second is when the Pushback Agent has other information that a DoS attack is in progress. For example, Pushback can be invoked by a router at the behest of a server directly connected to it. This is helpful in situations when considerable traffic is being sent to the server, but at a level not high enough for the ACC Agent at the router itself to invoke pushback.

5.2 Sending the Pushback Requests Upstream

When the Pushback Agent at the congested router invokes Pushback for an aggregate, it has to divide the rate limit for the aggregate among the upstream links. This requires that the Pushback Agent have some estimate of the amount of aggregate traffic coming from each upstream link. The upstream links sending only a small fraction of the aggregate traffic are termed as *non-contributing links*, and we call the other upstream links *contributing links*. Because one of the motivations of pushback is to concentrate the rate-limiting on the links sending the bulk of the traffic within the aggregate, the Pushback Agent does not send a *pushback request* to non-contributing links. The assumption is that if a DoS attack is in progress, the aggregate traffic from the non-contributing links is more likely to be legitimate traffic within that aggregate.

In the general case, contributing links do not all contribute the same amount of bad traffic. A link carrying more traffic belonging to the aggregate is more likely to be pumping in attack traffic. One of many possible algorithms, and the one used in our simulations, is to first determine how much traffic in the aggregate each link contributes. We then divide the desired limit L , reduced by the amount of traffic coming from non-contributing links, among the contributing links in a max-min fashion. For example, assume that we have three contributing links with arrival rates of 2, 5, and 12 Mbps, and that the desired limit, after the non-contributing traffic has been subtracted from it, is 10 Mbps. The limits sent to each of the three contributing links would then be 2, 4, and 4 Mbps respectively.

Congestion Signature
Bandwidth Limit
Expiration Time
RLS-ID
Depth of Requesting Node
Pushback Type

Figure 6: Contents of a pushback request

⁴The high drop rate implies that the router has not been able to control the aggregate locally by preferential dropping, in an attempt to encourage increased end-to-end congestion control.

After the Pushback Agent determines the limit to request from neighboring upstream routers, it sends a pushback *request* message to those routers. As shown in Figure 6, a pushback request contains the congestion signature characterizing the aggregate, the requested upper bound for the amount of traffic sent belonging to the aggregate, the time period after which the pushback request expires, the Rate-Limit Session ID (RLS-ID), the depth of the requester in the pushback tree, and the type of pushback. In our simulations the attack signature consists of the destination prefix or prefixes characterizing the aggregate. The RLS-ID is returned in the feedback messages (see Section 5.4) to enable the Pushback Agent to map the feedback to the corresponding pushback request. In the pushback tree, the depth of the root is zero, and a child's depth is one more than the depth of its parent. Depth information is useful in setting timers for sending feedback. The type of pushback influences the decision of an upstream router about whether to propagate pushback upstream. The router is more likely to propagate when the type corresponds to a malicious attack (*e.g.*, server-initiated pushback).

Note that the pushback request contains only a requested *upper* bound for the bandwidth to be given to the aggregate. If the upstream router itself becomes heavily congested, then it may give less bandwidth to the aggregate than the specified limit. Because the pushback request only specifies an upper bound, it will not end up shielding the aggregate from local congestion at the upstream router in the guise of rate limiting (see Section 4.3). But, the congested router could receive more than the desired amount of traffic in the aggregate if the non-contributing upstream neighbors (which were not sent pushback requests) start sending more traffic in the aggregate.

5.3 Propagating Pushback

On receiving a pushback request, the upstream router starts to rate-limit the specified aggregate just as it does for local ACC, using the rate limit in the request message. The router's decision whether to further propagate the pushback request upstream uses similar algorithms to those described in Sections 5.1 and 5.2 above.

When propagating a pushback request upstream, the destination prefixes in the congestion signature may have to be narrowed, to restrict the rate-limiting to traffic headed for the downstream congested router only. This is discussed in more detail in Appendix C.1.

5.4 Feedback to Downstream Routers

The upstream routers rate-limiting some aggregate in response to a pushback request send pushback *status* messages to the downstream router, reporting the total arrival rate for

that aggregate from upstream, along with the RLS-ID from the pushback request. The total arrival rate of the aggregate upstream is a lower bound on the arrival rate of that aggregate that the downstream router would receive if upstream rate-limiting were to be terminated. Because the upstream rate-limiting (dropping) may have been contributing to end-to-end congestion control for traffic within the aggregate, terminating the upstream rate-limiting may result in a larger arrival rate for that aggregate downstream. These pushback status messages enable the congested router to decide whether to continue the pushback. The timing of the pushback status messages is described in more detail in Appendix C.2.

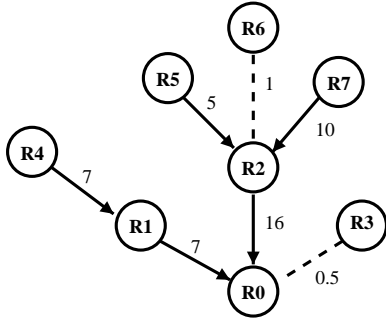


Figure 7: **Pushback status messages reporting the aggregate's arrival rate from upstream.**

The arrival rate reported in the pushback status message is the sum of the arrival rates in all the status messages received from upstream, plus the arrival rates from the upstream non-contributing nodes. For example, in Figure 7, $R0$ is the root of the pushback tree, shown by the solid lines. The labels for each solid line show the arrival rate estimate contained in the pushback status message. The dashed lines connect the non-contributing nodes that did not receive pushback request messages, and the labels show the aggregate's arrival rate as estimated by the downstream neighbor. From the pushback status messages, $R0$ can estimate the total arrival rate for the aggregate as 23.5 Mbps. If $R0$ were to terminate the rate-limiting upstream, and invoke an equivalent rate-limiting locally, this would be roughly the arrival rate that $R0$ could expect from that aggregate.

5.5 Pushback Refresh Messages

The Pushback Agent at the router uses soft state, so that rate limiting will be stopped at upstream routers unless refresh messages are received from downstream. In determining the updated rate limit in the refresh messages, the downstream router uses the status messages to estimate the arrival rate from the aggregate, and then uses the algorithms in Section 4.4 to determine the bandwidth limit. The arrival rates reported in the pushback status messages are also used by

the downstream router in determining how to divide the new bandwidth limit among the upstream routers.

6 Simulations with Pushback

This section shows a number of simulations using the NS [NS] simulator testing the effect of local ACC and pushback in a variety of aggregate-based congestion scenarios. Before going into the details of the simulations we introduce some informal terminology here that would help us in describing the simulations. For the scenarios with DoS attacks, the *bad* sources send attack traffic to the victim destination D , and the *poor* sources are innocent sources that happen to send traffic to the destination D when it is under attack. In other words, packets from the poor sources represent the unmalicious traffic in the congestion signature. For all of the scenarios, the *good* sources send traffic to destinations other than D .

6.1 A Simple Simulation

Figure 8 shows the topology for a simple simulation intended to show the dynamics of pushback. The good and the poor sources each send traffic generated by seven infinite demand TCPs. The bad source sends UDP CBR traffic, with the sending rate varied from one simulation to the next.

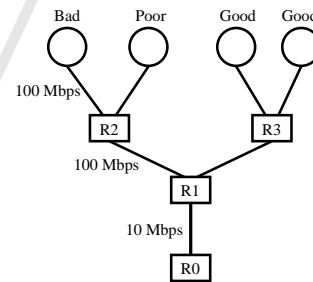


Figure 8: **The topology for a simple simulation.** $R1 - R0$ is the congested link.

The results of the simulation are shown in Figure 9. Each column of marks represents the results from a single simulation, with the x -axis indicating the sending rate of the bad source. When the bad source sends 8 Mbps or more, the drop rate at the output queue exceeds 10%, the configured value of p_{high} , and ACC and Pushback are initiated for the aggregate consisting of the bad and poor traffic. As a result of the rate-limiting, the arrival rate to the output queue is reduced, and the good traffic is protected from the bad.

For this scenario, the use of pushback is also effective in concentrating the rate-limiting on the bad traffic and protecting the poor traffic within the aggregate. The simulations with

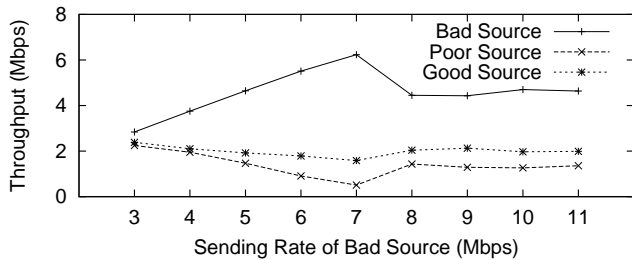


Figure 9: The effect of pushback in a small topology.

local ACC without pushback (not shown) produced approximately the same result for the good traffic; however, the poor source received almost no bandwidth in that situation. We would emphasize that these simulations do not pretend to use realistic topologies or traffic mixes, or to stress local ACC and pushback in difficult or highly dynamic environments; the simple simulations in this scenario are instead intended to illustrate some of the basic underlying functionality of local ACC and pushback.

6.2 DoS Attacks

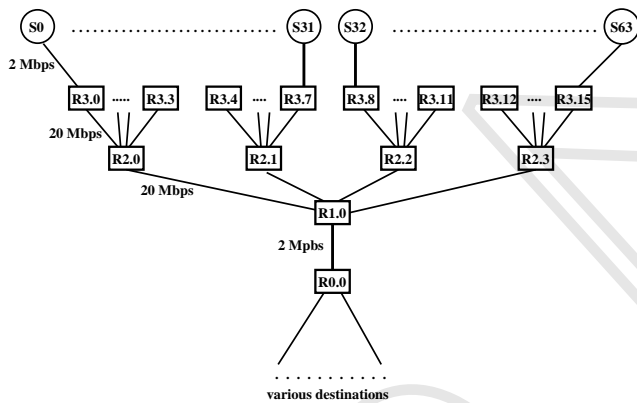


Figure 10: The topology used in simulations. $R1.0 - R0.0$ is the congested link

The simulations in this section illustrate local ACC and Pushback with both sparsely-spread and with highly diffuse DoS attacks. These simulations use the topology shown in Figure 10, consisting of four levels of routers. There is one router each in the bottom two levels and 4 and 16 routers, respectively, in the upper two levels. Except for the router at the lowest level, each router has a fan-in of four. The top-most routers are attached to four sources each. The link bandwidths are shown in the figure, and have been allocated such that congestion is limited to the access links at the top and bottom.

The first simulation scenario, with a sparsely-spread DoS attack, includes four bad sources, four poor sources, and

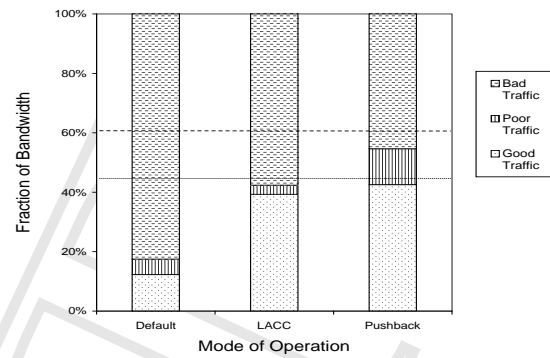


Figure 11: Bandwidth allocation at the congested link during a sparse DDoS attack.

ten good sources, randomly distributed among the 64 source nodes. Each of the four bad sources sends 1 Mbps of UDP CBR traffic, half the link capacity. The good and the poor sources send Web traffic, using the Web traffic generator in NS.

Figure 11 shows the results of these simulations. “Default” denotes a simulation with the router not doing any form of ACC, and “LACC” denotes the use of purely Local ACC measures. The two lines in the graph denote the quantity of good and poor traffic in the absence of any attack traffic. Both the LACC and pushback simulations bring down the bandwidth consumed by the attacker, leading to a significant bandwidth gain for the good traffic. However, as expected, LACC leaves the poor hosts starved because the congested router, $R1.0$, cannot differentiate between the poor and the bad traffic in the aggregate. We obtained similar results for simulations with different amounts of attack traffic.

The second simulation scenario, with a highly diffuse DoS attack, uses 32 bad sources, four poor sources, and ten good sources. In this scenario each of the 32 bad sources sends 0.125 Mbps of UDP CBR traffic, for the same total bad traffic as in the previous scenario. This setup is intended to simulate a DDoS attack where a large number of sources spread throughout the network are used to generate the attack traffic. Each bad source by itself generates a small amount of traffic, making it hard to detect such sources at their access links.

As Figure 12 shows, with diffuse attacks even pushback loses the ability to differentiate between the bad and the poor traffic, though it still reduces the bandwidth consumed by the bad sources. In fact, in an attack in which a lot of sources are used, an individual bad source might be generating less traffic than a valid poor source. When these bad sources are spread throughout the network, the attack looks more like a flash crowd, making it harder to distinguish between the bad and the poor sources. The bandwidth obtained by the good traffic with pushback goes slightly above the no-attack case (lower line) because of reduced competition from the poor

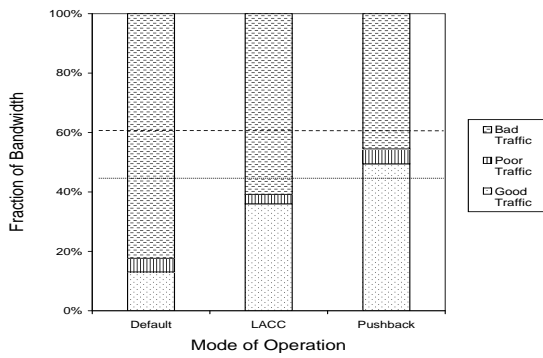


Figure 12: Bandwidth allocation at the congested link during a diffuse DDoS attack.

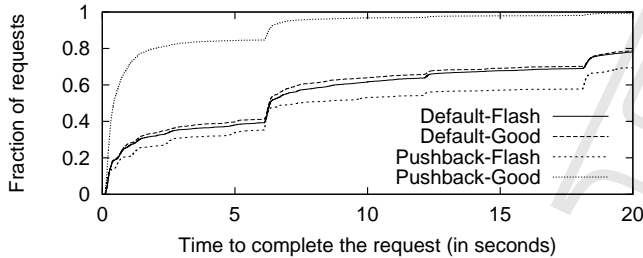


Figure 13: Time to complete a request during a flash crowd.

traffic.

6.3 Flash Crowds

This section shows simulations with flash crowds instead of DoS attacks, with the “flash” traffic from 32 sources sending Web traffic to the same destination. The good traffic comes from ten other sources sending Web traffic to various other destinations, accounting for about 50% link utilization in absence of any other traffic.

Figure 13 shows the distribution of the times to complete the transfers for the good and the flash traffic respectively in the Default and Pushback mode. The distribution for LACC mode (not shown) was similar to the Pushback one. With Pushback, 80% of the good transfers complete within a few seconds, compared to less than 40% completed in less than six seconds in the Default case. While the performance gain for the good traffic is significant, the degradation seen by the flash traffic is not that much. The time to complete a Web request can be directly correlated to the drop rate experienced. Drop rate for the good traffic comes down from a whopping 30% to just 6% ($p_{target}=5\%$) and that of the flash traffic goes up only by 3% to about 33%. Because the flash traffic is much more than the good traffic, even a slight increase in its drop rate frees up lot link capacity.

The hump around the 6-second mark represents short web transfers whose first SYN or SYN/ACK packet was lost, resulting in the transfer completing slightly more than six seconds later, after the retransmit timer expires. The magnitude and the location of the hump along the y-axis is a good indication of the packet drop rates in the network for that aggregate. Recall that LACC and Pushback are only invoked in scenarios of extreme congestion where the packet drop rate exceeds the configured threshold, set to 10% in our simulations, and at these levels of congestion a large fraction of transfers will have the first SYN or SYN/ACK packet dropped.

Though the graph did not show major differences between the transfer time distribution of web requests for LACC and pushback, the good traffic receives roughly 37% of the link bandwidth with LACC, compared to 50% with pushback. Because pushback rate-limits the bad traffic upstream, this leads to a decrease in the amount of bad traffic reaching the congested router, relative to LACC (absence of statistical multiplexing), which in turn enables more good traffic to go through. The transfer time distribution on the other hand is a function of the drop rate. Thus, more good traffic gets through with pushback than with LACC while keeping the same ambient drop rate at the output queue.

7 Discussion

7.1 When is Pushback Needed?

Pushback is not a panacea for flooding attacks. In fact, if not used carefully, it can make matters worse. This section discusses the advantages and disadvantages of adding pushback to aggregate-based congestion control.

7.1.1 Advantages of Pushback

Pushback concentrates rate-limiting on the malicious traffic within an aggregate during a DoS attack even in the presence of spoofed source addresses. The malicious traffic in the aggregate may come from a single link several hops upstream, but the downstream router may not be able to determine this, and the attacker could also disguise information by using spoofed source addresses. In this case, pushback using the destination prefix as a congestion signature can be effective in protecting the legitimate traffic within the aggregate. In contrast, local ACC would not be able to distinguish between the good and the bad traffic headed for the victim and would punish traffic coming from all directions equally.

Pushback also improves network utilization by preventing scarce upstream bandwidth from being wasted on packets that will be dropped downstream. Even if all routers are employing local ACC, information from a downstream router can

encourage an upstream router to increase the drop rate for an identified aggregate, thus saving bandwidth for other traffic.

7.1.2 Limitations of Pushback

Pushback may overcompensate, particularly when it is invoked for non-malicious events such as flash crowds, which eventually respond to congestion control and back off. If the overall demand from other traffic is reduced before the pushback refresh period expires (Section 5.5), then the upstream routers could unnecessarily drop packets from the high-bandwidth aggregate even when the downstream link becomes underutilized. (In Local ACC link underutilization is more easily avoided, as rate-limiting does not drop packets when the output queue is itself low.) We reduce the possibility of overcompensation (and lower link utilization) by calculating the rate-limit of an aggregate so that the total traffic coming to the congested router is still greater than the capacity of the congested link (see the discussion of p_{target} in Section 4.2). Performing some of the rate-limiting just in local ACC can also help to prevent overcompensation.

Pushback is less effective at blocking bad traffic when the attack is uniformly distributed across inbound links, as all traffic, good and bad, that falls within the aggregate is equally punished. Consider, for example, a reflector attack [Pax00] based on DNS [CER00]. If sufficiently many reflectors are used from all portions of the network, the aggregate bandwidth will swamp the victim's link. During such an attack pushback will not be able to differentiate between the good and the bad DNS traffic going to the destination, and will drop from both equally.

In some cases, the use of pushback can increase the damage done to legitimate traffic from a source close to the attacking host. As pushback propagates upstream towards the attack sources, the drop rate for the aggregate is increased. If pushback fails to reach a point where it can differentiate between the attack sources and the nearby legitimate traffic within the same aggregate, for instance, when the two sources are in the same edge network which is not pushback-enabled, the legitimate traffic at that point will share the same high drop rate as the attack traffic. This property of pushback could lead to potential DoS attacks in which the attacker's aim is to hinder a source from being able to send to a particular destination. To be successful, an attacker would need to launch the attack from a host close to the victim source. However, the ability to compromise a machine that shares a downstream bottleneck link with the victim enables many other forms of attack anyway.

7.2 Implementation and Operational Issues

In this section we talk about some of the implementation and operational issues that would need to be addressed before pushback could be deployed in the Internet.

The network bandwidth requirement for pushback messages is minimal. During each refresh round of each pushback session, on the order of a few seconds, one message is sent over a link in the pushback tree in each direction. Strong authentication of pushback messages is also not required since all pushback messages travel between directly connected routers.

The identification of aggregates can be done as a background task, so the processing power required to identify aggregates should not be an issue. However, the presence of a large number of rate-limited aggregates could pose a design challenge. When a packet arrives to the output queue, the router has to determine if that packet belongs to one of the rate-limited aggregates, and if so, place it in the correct virtual queue. The time required for this lookup may increase with an increasing number of aggregates. We do not expect the limitation on the number of rate-limited aggregates to be a problem, as we envision ACC and Pushback as mechanisms to be instantiated sparingly, in times of high congestion, for a handful of aggregates. But a deployed system needs to be robust against new attacks that could generate many rate-limited aggregates. One possible approach would be to use the routing table of the router for detecting membership in a rate-limited aggregate; however, this would restrict the definition of aggregates to destination prefixes.

The distribution of the rate-limit among upstream links depends on the downstream router's ability to estimate what fraction of the aggregate comes from each upstream link. We do not know if current routers are able to map a packet at the output queue to its incoming input interface. Inter-router technology will also affect the design of a pushback system. For point-to-point links, pushback requests are sent to the router attached to the contributing incoming interface. However, for routers joined by LANs, VLANs, or frame relay circuits, there are multiple routers attached to an interface. Similarly, some routers may only be able to determine from which line card traffic originated, rather than which port on the card. The downstream router in these situations might not be able to distinguish between multiple upstream routers. One way of dealing with both these problems is to send a *dummy* pushback request to all upstream neighbors. The dummy request is identical to the real request, but contains a very high bandwidth limit so that no dropping takes place in the upstream rate-limiter. The only impact of this request is that the upstream routers will estimate the arrival rate of the specified aggregate and report to the downstream router in status messages. These messages help the downstream router to send pushback requests with the appropriate rate-limits to contributing upstream routers.

In our simulations, we noted that “poor” hosts suffered more than “good” ones, primarily because the access routers could not push back further. The same situation could arise when pushback is incompletely deployed; a router’s upstream neighbor may not implement it. In that case, input rate-limiting would be useful. That is, a router that cannot send a pushback message upstream could impose a rate limit on traffic from that particular link that matches the aggregate description. It is unclear, however, if this is feasible in today’s routers.

The existence of layer 2 devices that do their own queuing—ATM switches, MPLS subnets, and the like—poses a separate challenge. The queue congestion can easily occur in the switches, rather than in routers; in this case, the switches would need to implement their own pushback mechanisms. Furthermore, the pushback requests would need to cross the layer 2/layer 3 boundary when talking to ordinary routers.

7.3 Policy Knobs

Like traffic engineering, ACC and Pushback are areas that will take significant guidance from local policy databases. This is unlike many other router mechanisms such as per-flow or FIFO scheduling, active queue management, or Explicit Congestion Notification, which are generally best with a modest number of policy hooks. It will take many simulations and a good deal of operational experience to get a better understanding of the right policies for ACC.

In principle, there is nothing stopping a pushback tree from extending across ASs. However, real networks make extensive use of policy databases in making decisions involving other ASs. There are issues related to trust in accepting a pushback request from a neighboring AS. Moreover, a pushback request might be in contradiction with a contractual obligation that says “provide transit to this much traffic”. In this case an edge router could discover conflicts and inform its parent (and the congested router). This distribution of limits along the pushback tree might be heavily policy-driven in such cases.

Even within an AS, it is not totally clear how to allocate rate-limits to different input streams. As discussed in Section 5.2, it seems fairly clear that aggregate traffic from non-contributing upstream links should not be rate-limited. However, there is no one best answer for how to allocate upstream rate-limit requests among different contributing links. Different choices here affect the ability of pushback propagation to reach the ultimate sources of congestion, and its ability to differentiate between malicious and unmalicious traffic. The issue becomes more complex when the contributing physical links are of different capacities. A mix of policy and operational experience might provide the right answer here.

7.4 Empirical Data on Traffic Behavior and Topologies

The starting question for ACC is what constitutes sustained congestion. What are the drop rates, over what period of time, that merit invoking ACC? We expect the answer to be different for different places in the network. For understanding this, it would help to have as much measurement data as possible on the pattern of packet drop rates at different routers in the Internet. How frequently do different routers have periods of sustained high congestion? How long does this sustained congestion last? And how often is it due to special events like flash crowds and DoS attacks, as opposed to the more diffuse congestion from hardware failures or routing changes for which ACC and Pushback would be less appropriate? More measurement data could also help in considering the issue of when to invoke Pushback in addition to local ACC. Sites such as the Internet Traffic Report [ITR] and the Internet Weather Report [IWR] have some data on packet drop rates, as well as reports on specific fiber cuts, flash crowds, and DoS attacks, but we are not aware of any systematic characterization and identification of the high-congestion periods at a specific router. As a further complication, past measurements are not necessarily good predictors of future conditions for such volatile occurrences as flash crowds and DoS attacks, either at an individual router or for the Internet as a whole.

The dynamic behavior of pushback is affected by the frequency and expiration times of pushback request messages. For robustness against dropped or corrupted pushback messages, the expiration times in pushback requests should be at least a small multiple of the refresh interval, the interval between successive request messages. The cost of lower-frequency refreshes and larger expiration times would be the possibility of continuing rate-limiting upstream longer than desired, as a result of a lost refresh message that would have increased the requested rate-limit. However, because pushback is invoked only on rare occasions, and not as a substitute for normal congestion control or traffic engineering, this potential cost of continuing rate-limiting upstream is not a major problem.

As noted before, the effectiveness of Pushback in differentiating between malicious and non-malicious traffic within an aggregate during a DoS attack is dependent on the distribution of attack sources and the paths that the attack traffic takes to reach the congested router. Pushback is more likely to be effective in protecting the non-malicious traffic when the branching factor of the attack tree is small, and pushback can effectively concentrate the rate-limiting on the attackers. More information of typical attack topologies from specific past DoS attacks, analytical topology models, or Internet topology databases would help us to evaluate the potential effectiveness of Pushback in protecting non-malicious traffic within an aggregate.

7.5 Other Uses of Pushback

A router could send dummy pushback requests with high rate-limits just to find out what fraction of the traffic within the specified aggregate is coming from which portion of the network. This could be particularly useful when the specified aggregate is the aggregate consisting of all traffic destined to the congested router. In a time of stable, diffuse congestion, this would give the congested router the ability to determine if most of the diffuse congestion was coming from a particular edge router. Even in the absence of congestion this mechanism could be used to collect data on what edge router feeds what amount of a particular aggregate. This information is highly valuable for purposes of traffic engineering.

8 Conclusions

Most so-called network security problems are nothing of the sort. Rather, they are host vulnerabilities, albeit in applications that use the network. As long as these vulnerabilities can be exploited, it is possible to launch DoS attacks on the network or the hosts connected to it without fear of consequences. Flooding-style DoS attacks *are* a network problem, and cannot be defeated by host-specific mechanisms. Furthermore, no software fixes can prevent flash crowds. Although ISPs can and do engineer their networks for ordinary overloads, the network should be able to survive sudden, massive congestion caused by aggregates generated by either attacks or flash crowds.

We have proposed both local and cooperative mechanisms for aggregate-based congestion control to answer this need and believe that they are very promising directions. There are a range of open issues, and future work will include more thorough investigations of the underlying dynamics and possible pitfalls of these mechanisms. As part of this work, we are in the process of implementing an experimental testbed using software routers.

Acknowledgments

The original idea for pushback came from an informal DDoS research group (Steven M. Bellovin, Matt Blaze, Bill Cheswick, Cory Cohen, Jon David, Jim Duncan, Jim Ellis, Paul Ferguson, John Ioannidis, Marcus Leech, Perry Metzger, Robert Stone, Vern Paxson, Ed Vielmetti, and Wietse Venema).

References

[Aka] Akamai home page. <http://www.akamai.com>.

- [Art97] Artsci.net Web Pages: Dostrack. <http://www.artsci.net/~jlinux/security/dostrack>, October 1997.
- [BBC⁺98] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An Architecture for Differentiated Services. RFC 2475, December 1998.
- [BC00] Hal Burch and Bill Cheswick. Tracing Anonymous Packets to Their Approximate Source. In *Usenix LISA*, December 2000.
- [Bel00] Steve M. Bellovin. ICMP Traceback Messages. Internet Draft: draft-bellovin-itrace-00.txt, March 2000.
- [CER96] CERT Web Pages: CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks. <http://www.cert.org/advisories/CA-1996-21.html>, September 1996.
- [CER98] CERT Web Pages: CERT Advisory CA-98.01 "smurf" IP Denial-of-Service Attacks. <http://www.cert.org/advisories/CA-98.01.smurf.html>, January 1998.
- [CER00] CERT. Cert incident note in-2000-04, 2000. http://www.cert.org/incident_notes/IN-2000-04.html.
- [Cis98] Cisco Web Pages: Committed Access Rate. <http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/car.htm>, February 1998.
- [CSZ92] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network Architecture and Mechanism. In *ACM SIGCOMM*, 1992.
- [Dav00] Brian D. Davidson. Content delivery and distribution services. <http://www.web-caching.com/cdns.html>, August 2000.
- [DBCP97] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *ACM SIGCOMM*, September 1997.
- [DFS01] Drew Dean, Matt Franklin, and A. Stubblefield. An algebraic approach to ip traceback,. In *Proceedings of NDSS '01*, February 2001.
- [Dig] Digital Island home page. <http://www.digitalisland.com>.
- [DKS89] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *ACM SIGCOMM*, 1989.

- [FF97] Sally Floyd and Kevin Fall. Router mechanisms to support end-to-end congestion control. Technical report, LBL, February 1997.
- [FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, Vol. 1(4):pp. 397–413, August 1993.
- [FJ94] Sally Floyd and Van Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Transactions on Networking*, Vol. 2(2):pp. 122–136, April 1994.
- [FJ95] Sally Floyd and Van Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, Vol. 3(4):pp. 365–386, August 1995.
- [FS00] Paul Ferguson and Daniel Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.
- [Gar00] Lee Garber. Denial-of-Service Attacks Rip the Internet. *IEEE Computer*, vol. 33(4):pp. 12–17, April 2000.
- [ICS00] ICSA Web Pages: Alliance for Internet Security. <http://www.icsa.net/html/communities/ddos/alliance/guide.shtml>, 2000.
- [ILS99] A. S. Induruwa, P. F. Linington, and J. B. Slater. Quality of Service measurements on SuperJANET - the UK academic information highway. In *Proc INET'99*, June 1999.
- [ITR] The Internet Traffic Report. <http://www.internettrafficreport.com/>.
- [IWR] The Internet Weather Report. <http://www.mids.org/weather/>.
- [LM97] Dong Lin and Robert Morris. Dynamics of Random Early Detection. In *ACM SIGCOMM*, 1997.
- [MF00] Ratul Mahajan and Sally Floyd. Controlling High-Bandwidth Flows at the Congested Router, November 2000. <http://www.aciri.org/red-pd/>.
- [NS] NS Web Page: <http://www.isi.edu/nsnam>.
- [Par96] Craig Partridge. Locality and Route Caches. NSF Workshop on Internet Statistics Measurement and Analysis, February 1996.
- [Pax00] Vern Paxson. An analysis of using reflectors to defeat DoS traceback, August 2000. <ftp://ftp.ee.lbl.gov/vp-reflectors.txt>.
- [PPP00] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis. CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *IEEE INFCOM*, 2000.
- [SAN00] Egress Filtering. SANS Web Pages, February 2000. <http://www.sans.org/y2k/egress.htm>.
- [SDS00] Dan Schnackenberg, Kelly Djahandari, and Dan Sterne. Infrastructure for intrusion detection and response. In *Proceedings of the DARPA Information Survivability Conference and Exposition 2000*, March 2000. ftp://ftp.tislabs.com/pub/IDIP/DISCEX_IDR-Infrastructure.pdf.
- [SP01] Dawn Song and Adrian Perrig. Advanced and authenticated techniques for ip traceback. In *Proceedings of Infocomm 2001*. IEEE, 2001. <http://paris.cs.berkeley.edu/dawn-song/papers/iptrace.ps>.
- [SSZ98] Ion Stoica, Scott Shenker, and Hui Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *ACM SIGCOMM*, 1998.
- [Sto00] Robert Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *9th Usenix Security Symposium*, August 2000.
- [SV95] M. Shreedhar and George Varghese. Efficient Fair Queuing using Deficit Round Robin. In *ACM SIGCOMM*, 1995.
- [SV98] V. Srinivasan and George Varghese. Faster IP Lookups using Controlled Prefix Expansion. *ACM Transactions on Computer Systems*, November 1998.
- [SV00] Sandeep Sikka and George Varghese. Memory-Efficient State Lookups with Fast Updates. In *ACM SIGCOMM*, August 2000.
- [SWKA00] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical Network Support for IP Traceback. In *ACM SIGCOMM*, August 2000.
- [WVTP97] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable High Speed IP Routing Lookups. In *ACM SIGCOMM*, September 1997.

A Preferential Dropping as a Rate-limiting Mechanism

Section 4.3 described the virtual queue as the mechanism used for rate-limiting aggregates. This section describes an alternate mechanism, preferential dropping, and contrasts this with the virtual queue.

When preferential dropping is used as the rate-limiting mechanism, the aggregate's arrival rate r_a is estimated. Given the specified bandwidth limit r_l for the aggregate, the rate-limiter drops each arriving packet in the aggregate with probability $1 - r_l/r_a$, so that the expected rate of traffic leaving the rate-limiter is the desired limit r_l . For example, if the arrival bandwidth r_a is 10 Mbps and the target bandwidth r_l is 7.5 Mbps, the incoming packets will be dropped with a probability $1 - r_l/r_a = 1/4$. When $r_l \geq r_a$, no packet will be dropped by the rate limiter.

Next we compare the preferential dropping mechanisms with the virtual queue. The preferential drop mechanism and the virtual queue both test arriving packets, forwarding some and dropping others, so that the forwarded packets are roughly limited to the specified bandwidth. However, there are behavioral differences between the two mechanisms.

Both mechanisms must maintain some state: the preferential drop mechanism maintains state for rate estimation, and the virtual queue maintains state for the virtual queue simulation.

The virtual queue is more sensitive than the preferential drop mechanism to the short-term burstiness of the arrival process. For example, once the virtual queue is full, the virtual queue is likely to drop a burst of incoming packets, while the preferential dropping mechanism spreads out its drops more evenly.

The preferential drop filter does not guarantee that its output is no greater than the specified bandwidth; the estimate of the arrival rate is not necessarily accurate, and, because each packet is dropped with a certain probability, only the expected behavior of the preferential drop filter can be guaranteed. In contrast, the virtual queue precisely controls the exit rate, as a function of the queue size and service rate of the virtual queue. The preferential drop filter could keep dropping for a while after the arrival rate of the aggregate has been reduced, because it might take some time for the estimate of the arrival rate to reflect the actual reduction in the arrival rate. In contrast, the virtual queue will respond promptly to a slow-down in the arrival rate of the aggregate.

A deployment advantage of virtual queues is that they are already available in commercial routers [Cis98].

B Local ACC

This section gives the pseudocode for the algorithms used for Local ACC. We must note that it does not specify the algorithms in minute details; for this, the only reference, at the moment, is the pushback code in the NS simulator.⁵ The sole purpose of presenting this skeletal pseudocode here is to give the reader a sense of underlying algorithms. Figures 14 - 18 contain the pseudocode.

Objects Used in ACC

OQ Output Queue
RL Rate Limiter
ACC ACC Agent

Parameters

p_{high} drop rate to trigger aggregate-based congestion control
 p_{target} target ambient drop rate at OQ
 K time period for checking high drop rate
 $T_{refresh}$ time period for reviewing the limit imposed on the aggregates
 $MaxClusters$ maximum number of aggregates to rate-limit simultaneously

Variables

$OQ.R_{estimate}$ arrival rate estimate
 R_{target} (Link BW)/(1 - p_{target})
 $DropLog$ drop history
 $LowerBound$ Arrival rate of biggest non-rate-limited aggregate (dynamically updated)

Figure 14: Some definitions used in the pseudocode

C Pushback: Implementation Details

C.1 Propagating Pushback Upstream

When propagating a pushback request upstream, the destination prefixes in the congestion signature might have to be narrowed, to restrict the rate-limiting to traffic headed for the downstream congested router only.

⁵The pushback code in the NS simulator is in “ns/pushback”, and the validation test is in file “ns/tcl/test/test-all-pushback”.

```

/* invoked on timer expiry */
timeout()
1. Estimate  $p$ 
   (the drop rate at the queue)
2. if ( $p > p_{high}$ )
   ACC.identify_aggregate()
   else
   ACC.updateLowerBound()
3. Reset DropLog
4. Set up timer with  $K$  second delay.

/* invoked by RL for
packet insertion into OQ */
enqueue(pkt)
1. Update  $R_{estimate}$ 
2. Check if pkt is to be dropped
   (based on the queue management method).
3. if (dropped)
   Log pkt in DropLog
   else
   Insert pkt in queue.

```

Figure 15: **Methods of Output Queue**

Consider the following scenario. Suppose the congested router X identifies a certain aggregate A with dst prefix 128.95.0.0. X will ask its upstream router Y (among others) to rate-limit traffic from aggregate A (128.95.0.0). However, Y cannot use the same specification directly because while Y could be forwarding 128.95.1.0 to X, it might not be forwarding the rest of 128.95.0.0 to X. If Y (and routers upstream of Y) started rate-limiting all of 128.95.0.0, the network would drop traffic which would not have reached the congested router.

To avoid this unnecessary packet-dropping, it is important that Y look at its routing table to find which prefixes within 128.95.0.0 are forwarded to X. Y has to check all extensions of the given prefix in the routing table. For example, if X specifies a prefix bbbb, Y would check bbbb0 and bbbb1. Any branch that does not exist is covered by the request, so no further searching is needed. If the branch exists, the algorithm is applied recursively. This check is reasonably cheap. Existing IP lookup schemes are based on either multibit tries or binary search of hash tables [SV00]. Both multibit tries [DBCP97, SV98] and hashing schemes [WVTP97] enable fast prefix lookups (in fact they lookup prefixes extracted from the destination address). Prefix lookups cannot be done in caches as caches usually store complete addresses, but this is not a limitation as even in the normal forwarding process cache misses are not an uncommon occurrence [Par96]. It should also be noted that most of the times upstream routers

```

/* invoked when a new packet arrives */
enqueue(pkt)
1. Check if pkt belongs to a
   rate-limited aggregate.
2. if (yes) {
   Update the arrival rate estimate
   of the aggregate
   Check if packet needs to be dropped
   (based on virtual queue state)
   if (!dropped)
   OQ.enqueue(pkt)
   }
   else
   OQ.enqueue(pkt)

/* invoked by ACC to install new
aggregates for rate-limiting */
limit(Listagg, num, L)
1. N = Number of rate-limited
   aggregates
2. Remove aggregates that are already
   being rate-limited from  $List_{agg}$  and
   change their limit to  $\min(\text{oldLimit}, L)$ 
3. Pick the top sending  $MaxClusters$ 
   aggregates from already rate-limited
   aggregates and  $List_{agg}$ .
   (Normally, the total in two lists would
   be less than  $MaxClusters$ , so all of them
   will be picked).
4. Install filters for new ones with
   limit L.
5. Release (after some time) the old
   aggregates which were not picked.

```

Figure 16: **Methods of Rate Limiter**

will not have a longer prefix in the routing table, because it is not very common for an upstream router to have longer prefixes than the downstream one (longer prefixes tend to occur closer to destinations).

C.2 Pushback Request Messages

Pushback status messages are sent one hop downstream. Leaf nodes use timers to send status messages. A non-leaf node sends status message when it has received status messages from all its children. In case a child fails to send a status message in a round, the parent router will eventually timeout and send the status message downstream using the last value received from this child or its own estimate. The timer val-

```

/* invoked to get a sorted list of
aggregates. this method uses
the definition of aggregates.
the steps shown below are for
destination based clustering
mentioned in the text (§4.1) */
get_clusters()
1. High32 = List of destinations
with more than mean number of
drops in DropLog
2. High24 = List of 24 bit prefixes
in High32
3. Clusters0 = List of prefixes
after merging close prefixes
in High24
4. Clusters1 = List of prefixes
longer than those in Clusters0 but
containing most drops in the sub-tree
5. return sorted Clusters1
(decreasing number of drops)

/* invoked by OQ to when drop rate
goes above  $p_{high}$  */
identify_aggregate()
1. Clusters = get_clusters()
2. Estimate arrival rate of each
prefix in Clusters using
agg_arr=( $OQ.R.estimate$ )*
(agg_drops/total_drops)
3.  $R_{excess} = OQ.R.estimate - R_{target}$ 
4. i=1, done=0, sum_rate=0
5. while not done:
sum_rate+=Clusters[i].agg_arr
L = (sum_rate -  $R_{excess}$ )/i
if (L  $\geq$  Clusters[i+1].agg_arr)
done=1, break
else
if (i=MaxClusters)
break
else
i++
6. RL.limit(Clusters[1..i],L)
/* this lowerBound would be applicable to
already identified aggregates also during
the next refresh */
7. LowerBound=Clusters2[i+1].agg_arr

```

Figure 17: Methods of the ACC Agent

ues for status messages are set based on the node's depth in the Pushback tree so that failure of one node does not trigger

```

/* invoked every  $T_{refresh}$  seconds to
review the limit on aggregates */
refresh()
1. Merge related prefixes
(E.g., with close prefix)
2. N = Number of rate-limited
aggregates
3.  $Limit_{total}$  = total limit on
aggregates
4.  $Arr_{total}$  = total arrival rate of
aggregates
5.  $R_{incoming} = OQ.R.estimate - Limit_{total} + Arr_{total}$ 
6.  $R_{excess} = R_{incoming} - R_{target}$ 
7. L = ( $Arr_{total} - R_{excess}$ )/N
8. If (L < LowerBound)
L = LowerBound
9. foreach A in (List of Aggregates)
if (A.arr < L)
/* actual release happens after some time
if A continues to have a low arrival
rate */
release A
else
change limit of A to L
10. Set up timer for next refresh

/* invoked to update lowerBound */
update_lower_bound()
1. Clusters = get_clusters()
2. lowerBound = arrival rate of first
aggregate which is not rate-limited.
3. average lowerBound exponentially.
(to get rid of temporary fluctuations)

```

Figure 18: Methods of the ACC Agent (contd.)

timeouts in all its ancestors and force them to use stale values.

C.3 Pushback Refresh Messages

On receiving the Pushback refresh the upstream routers update the expiration time for the rate-limit session, the limit imposed on the aggregate and the aggregate specification (if it has changed). Timers, whose value depends on the depth in the tree, are set for status messages at this point. Routers which are not leaves in the Pushback tree send a refresh message further upstream after dividing the limit and checking with the routing table as to what prefix should be sent.