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*

Abstract

The current Internet infrastructure has very few built-in protection mechanisms and is therefore vulnerable to attacks and failures. In particular, recent events have illustrated the Internet's vulnerability to both denial of service (DoS) 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 DoS 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, provide relief from flash crowds and flooding-style DoS attacks.

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 [6, 16, 24, 18] on mechanisms to cope with such *ill-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). Second, as was seen on the transatlantic links a few years ago, persistent overloads can also be due to a general excess of traffic [15]. While better active queue management techniques [11] 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 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 DoS attacks occurred in February 2000 to considerable media attention, resulting in higher packet loss rates in the Internet for several hours [14]. DoS attacks have also been directed against network infrastructure rather than against individual web servers [19].

Flash crowds occur when a large number of users try to access the same server simultaneously. Apart from overloading 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 Secret webcast [2] and during the

*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, and shenker@aciri.org

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)[1] 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, an *aggregate* as a collection of packets from one or more flows that 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 (Local ACC), 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. Pushback can prevent upstream bandwidth from being wasted on packets that are only going to be dropped later on in the network. In addition, for a DoS attack, if the attack traffic is concentrated at a few incoming links upstream, then pushback can be effective in protecting other traffic within the aggregate from the attack traffic.

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. In Section 3 we describe some related work done to tackle the problem of DoS attacks and flash crowds. Section 4 describes the local component of 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 issues related to ACC.

2 Overview of ACC

This section gives 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. They are then explored in detail in §4 and §5.

We can think about an ACC mechanism running in a router (or possibly in an attached device) 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? When do I ask upstream routers to stop?

Each of these questions requires an algorithm for making the decision. Each is also a natural point to inject *policy*

[1] We note that the term "ACC" has been used in different contexts to denote "Active Congestion Control" and "ACK Congestion Control".

considerations into the decision making. The space of possible policies (e.g., who to treat better than whom, who 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. A small jitter will be applied to the monitoring interval, both to avoid synchronization effects [12] and to resist an attacker intent on predicting the response patterns of ACC in the presence of a DoS attack. Maintaining some longer-time history of the packet drop rate could help to detect the intermittent periods of heavy congestion that could result from DoS attacks.

When serious congestion is detected, the router attempts to identify the aggregate(s) 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 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 [11]. 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.

That the high bandwidth aggregates are in fact responsible for congestion is an assumption in our scheme. There are links in the network that are dominated by a particular aggregate(s), in the normal case. The ISP can use policy if it wants to protect such aggregates, resulting in ACC mechanisms looking for other aggregates or rate-limit these high bandwidth aggregates only when they exceed their policy defined limits. A possibility we have not explored yet is the use of history for identification.

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 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 (§4.2).

A more Draconian measure, like completely shutting off or imposing a very low bandwidth limit for identified aggregates, is not taken because of two reasons. First, the aggregate can be a flash crowd. Second, even if the aggregate is from a DoS attack, the congestion signature of the attack traffic will usually contain some innocent traffic too.

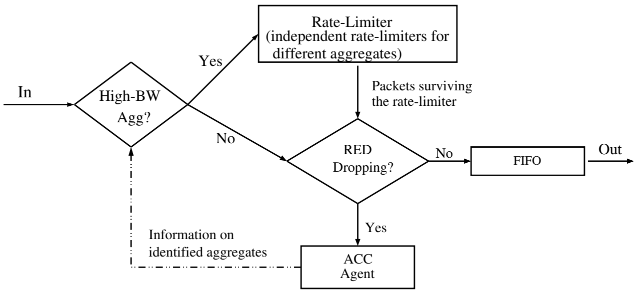

Figure 1: **The rate-limiting architecture.**

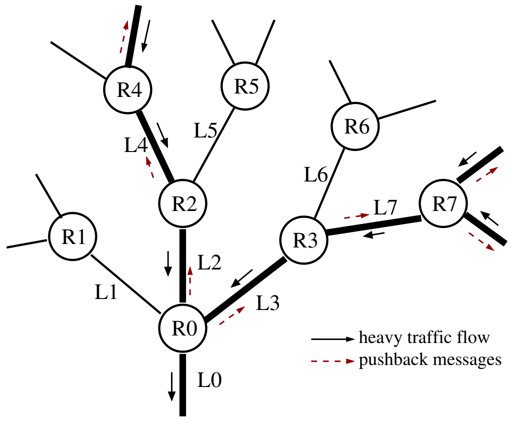

Figure 2: **Illustration of pushback**.

Figure 1 shows the rate-limiting architecture. There is 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. 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. In contrast, the rate-limited aggregates would get preferential treatment if they were allocated a fixed bandwidth share irrespective of the general congestion levels at the output queue.

We next turn to the possibility of using pushback to control an aggregate. But first, a more detailed description of pushback. 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 traffic for that aggregate.[2] Routers receiving these messages can recursively propagate pushback upstream (closer to the sources). 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. 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.

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. 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 [7]. Pushback 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.

The question "when to invoke pushback" is dealt with in §5.1. Briefly, pushback is useful when the router cannot find a narrow enough congestion signature (to minimize collateral damage), or when it is dealing with traffic that is known to be malicious (through some other information or by observing that traffic does not respond to increased drop-rate). Pushback can also be initiated by an overloaded server so that, in cases where a DoS at-

[2]Clearly, pushback messages require authentication, lest they provide a powerful denial-of-service mechanism themselves!

tack was not causing congestion but was overloading a server, the benefits of pushback would still be available. In addition, the decision as to when to use pushback is likely to have a large policy component, which we do not address in this work.

The last question posed at the beginning of this section was: "when do I stop?" For Local ACC, the answer is simple: the router continues to monitor its congestion. 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 motivates the need for *feedback* messages that the upstream routers send out reporting on how much traffic from an aggregate they are still seeing (§5.4).

3 Related Work

Two common mechanisms to counter DoS attacks are ingress filtering [7] and traceback [22, 1]. ACC is orthogonal to both of them; the focus of ACC is neither to stop the attacks (ingress filtering tries to stop attacks that use source address spoofing) nor to find the sources of these attacks (traceback), but to minimize the damage done by high-bandwidth aggregates. 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.[3]

Schnackenberg et al.[23] suggest active control of infrastructure elements. Thus, a firewall or Intrusion Detection System (IDS) that detected a DoS attack could request that upstream network elements block the traffic. The paper defines a protocol to express interaction between routers. It does not deal with mechanisms to identify or rate-limit malicious traffic, a subject of our work.

Web-caching infrastructures, content distribution networks (CDNs), and multicast are powerful mechanisms for preventing flash crowds from congesting the network. However, even a combination these techniques may not be sufficient to completely prevent network congestion from flash crowds. For example, flash crowds could occur for traffic not carried by 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 that witnesses an unprecedented "success" for which neither it nor the related infrastructure is prepared.

Flow-based congestion control is related to ACC but operates at a different granularity. This includes per-flow scheduling mechanisms like Fair Queuing [6], and mechanisms that use preferential dropping to approximate per-flow scheduling [24] or to protect conformant flows from misbehaving flows [9, 16, 18]. However, flow-based congestion control is not a solution for aggregate-based congestion, since an aggregate could be composed of many flows which are low-bandwidth individually. CBQ [13] is a class-based scheduling mechanism in which aggregates can be limited to a certain fraction of the link bandwidth during congestion. However, CBQ is discussed largely for fixed definitions of aggregates, and does not include mechanisms for detecting particular high-bandwidth aggregates during congestion.

4 Local ACC

We now describe the architecture and the algorithms used by the router to detect and control high-bandwidth aggregates. Pseudocode for the algorithms can be found in [17]. This section focuses on Local ACC, and the next on pushback.

Local ACC can be broken down into detection and control. In Figure 1, the ACC Agent is responsible for identifying 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, and might not even be on the same machine. Packets arriving to the output queue are checked to determine if they belong 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

[3] The 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.

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, random samples from the output queue can be used in the identification process.

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 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 (not taking 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 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, an aggregate could be defined only in terms of the protocol field or port number; all DNS packets, for instance. However, almost all DoS attacks and flash crowds have either a common source or a common destination prefix. As a result, we expect that most aggregate definitions will be based on either a source or destination address prefix. As is discussed later in the paper, pushback is invoked only for aggregates whose definition includes a destination address prefix.

We present a technique to identify high-bandwidth aggregates based on the destination address. The same technique could be used to identify aggregates based on the source address (though we acknowledge that source addresses cannot necessarily be trusted). This is only one of many possible algorithms for identifying high-bandwidth aggregates; more accurate and flexible algorithms are a subject of further research. We would note that more complex definitions of aggregates would require an appropriate language for expressing the aggregate definition and for passing the aggregate definition to upstream routers during pushback.

The identification technique presented below was designed with the observation that most Web sites operate in a small range of IP addresses[4].

Based on the drop history[5] (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 would give 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 also be formed for sites with spaced-out IP addresses. 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.

Since access links have much less capacity than backbone links, they are more likely to be congested during DoS attacks and flash crowds. The identification of high-bandwidth aggregates is easier in such cases. For instance, the aggregates for the congested router could

[4]Use of CDNs can result in a flash crowd near many caches; all routers that get congested will invoke Local ACC independently. Attacks, on the other hand, are likely to use IP addresses of the primary installation.

[5]With active queue management scheme (like RED) that distributes drops fairly, drops can be considered random sample of incoming traffic [10].

correspond to prefixes present in its routing table.

We note that different aggregates have quite different definitions, even in such basic characteristics as the number of IP addresses included in each destination-based aggregate. Thus, we reiterate that the notion of max-min fairness among aggregates is not viable as a general policy objective.

4.2 Determining the Rate Limit for Aggregates

Using the list of high-bandwidth aggregates obtained above during a period of high congestion, the ACC Agent determines if any aggregates should be rate-limited, and if so, what the rate-limit should be. The ACC Agent has a sorted list of aggregates, starting with the aggregate with the most drops from the drop history. The ACC Agent calculates the total arrival rate at the output queue, and uses this and the drop history to estimate the arrival rate from each aggregate over the most recent K seconds.

The ACC Agent next calculates R_{excess}, the excess arrival rate at the output queue. This 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}, in the worst case.

The procedure next determines the minimum number of aggregates that could be rate-limited to sufficiently reduce the total arrival rate. One constraint is that the rate-limit for rate-limited aggregates must be greater than the arrival rate of the largest non-rate-limited aggregate. A second constraint is that the total number of rate-limited aggregates must be at most $MaxSessions$.

If the ACC Agent has determined that it can rate-limit the i top aggregates, it next computes the rate-limit L to be applied to each aggregate. The limit L is computed such that

$$\Sigma_{k=1}^{i}(Aggregate[k].arr - L) = R_{excess},$$

where $Aggregate[k].arr$ is the arrival rate estimate of aggregate k.

The two constraints listed above ensure the L is less than the arrival rate estimate of $i+1$-th aggregate, and that i is at most $MaxSessions$. Ideally, the Local ACC mechanisms should not rate-limit *any* aggregate during times of undifferentiated congestion caused by under-provisioned links or hardware failures. In the absence of effective methods for distinguishing between aggregate-based and undifferentiated congestion, we use the upper bound $MaxSessions$ on the number of aggregates that are rate-limited simultaneously. With better understanding of the traffic composition and behavior during DoS attacks and flash crowds, we can tune the Local ACC mechanism such that it does not identify any aggregate in times of undifferentiated congestion.

In the presence of policy constraints, the computation of the rate-limit L would have to be modified slightly. For instance, the router could be configured never to rate-limit a specified aggregate to less than B Mbps. Such policy level decisions have to be honored in the rate limit calculation.

The ACC Agent revisits its rate-limiting decisions periodically, revising the rate limit L, and determining if some aggregate no longer needs to be rate-limited. The ACC Agent measures the arrival rate for rate-limited aggregates, so for the refreshes, the ACC Agent has more precise information about these arrival rates. 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. There is no harm done if rate-limiting continues for some time after the DoS attack or flash crowd has subsided, because the rate-limiter only drops packets if the arrival rate is more than the specified limit. However, some care is needed that the rate-limit for an aggregate does not change abruptly as another aggregate is added or removed from the list of rate-limited aggregates.

4.3 Rate-limiter

The rate-limiter is responsible for classifying packets, rate-limiting those 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 Figure 1 and describes a mechanism for implementing the rate-limiter.

Because it sits in the forwarding fast path, the rate-limiter needs to be light-weight and efficient to implement. 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. As noted earlier in §2 rate-limited aggregates are never 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.

4.3.1 Virtual Queue

In this section we discuss the virtual queue, the mechanism that we use for rate-limiting. 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, and packets that would have been queued at the virtual queue 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 implemented 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.

4.3.2 Narrowing the Congestion Signature

In the discussion above, the aggregates identified by the ACC Agent are based only or source or destination addresses. In fact, the rate-limiter can do more sophisticated narrowing of the congestion signature that, in times of specialized attacks, can result in dropping more of the attack traffic within the aggregate. For example, a suitably sophisticated ACC Agent would detect a more specific dominant signature within the aggregate, based on other packet characteristics (such as port number or ICMP type code), and drop more heavily from this subset. Narrower rate-limiting could be achieved by placing another virtual queue, with a smaller service rate, in front of the aggregate's virtual queue.

This hierarchical rate-limiting is safe in scenarios where attacker frequently changes her attack signature, as the total bandwidth available to the aggregate is still bound. Such specialized rate-limiting can be very useful in cases of attacks like the SYN attack [3] or the smurf attack [4].

One might perhaps argue that during flash crowds the routers should do some form of flow-aware rate-limiting, for example, dropping more heavily from SYN packets

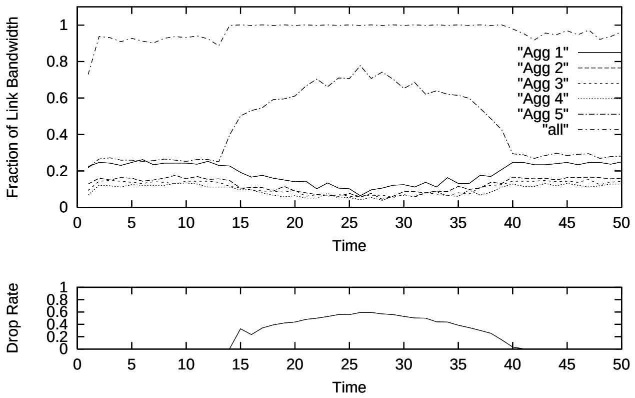

Figure 3: **A simulation without ACC.**

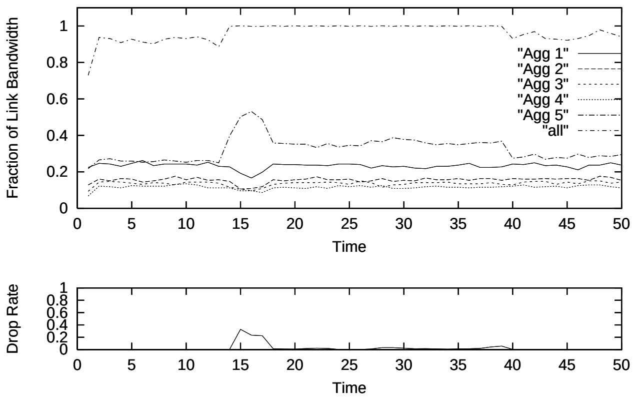

Figure 4: **The same simulation with Local ACC.**

to provide better service to connections that manage to get established. However, this can be dangerous if applied for a DoS attack rather than a flash crowd. The attacker could simply send packets in the category being favored by flow-aware rate-limiting (TCP data packets in the above example). Flow-aware rate-limiting is different from narrow rate-limiting mentioned above. While the latter punishes the dominant (relative to usual levels) packet type in the aggregate, the former favors a particular packet type, a strategy that can be gamed.

4.4 Simulations

We use a simple simulation to illustrate the effect of Local ACC. Figure 3 shows a simple simulation without ACC, with six aggregates, each composed of multiple CBR flows, with the sending rate of the fifth aggregate varying over time. Because this simulation is of CBR flows, rather than of flows using end-to-end congestion control, it has very simple dynamics; its purpose is to illustrate the underlying functionality of Local ACC.

The two graphs in Figure 3 show that, without ACC, the high-bandwidth aggregate is able to capture most of the link bandwidth. The bottom graph of Figure 3 shows the ambient packet drop rate in the output queue. At time 13 the sending rate of the fifth aggregate gradually increases, increasing the drop rate and decreasing the bandwidth received by the other four aggregates.

Figure 4 shows the same simulation repeated with Local 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 4 shows the ambient drop rate in the output queue, but does not show the drop rate in the rate-limiter for the fifth aggregate.

5 The Pushback Mechanism

Section 4 described Local ACC; let us now discuss the pushback mechanism in detail. Pushback for an aggregate can be visualized as a tree[6], 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 remains high for several seconds (because the arrival rate for the aggregate remains much higher than the limit imposed on it).[7] The second is when the Pushback Agent has other information that a DoS attack is in progress. In some cases packet drop history can help the router differentiate between DoS attacks and flash crowds. For instance, if most of the packets within the aggregate are destined for a notorious UDP port, the router can be fairly certain that it is witnessing a DoS attack and not a flash crowd. Another source of information can be the downstream server[8] itself. For example, pushback could be invoked by a router at the behest of a server directly connected to it, if allowed by the policy at the router. This would also be helpful to the server in situations when considerable traffic is being sent to the server, but at a level not high enough for the ACC Agent at the adjacent router to invoke Local ACC or 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 on the contributing links is more likely to include the attack traffic, while the aggregate traffic on the non-contributing links is more likely to be legitimate traffic.

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

[6] It would be a graph with multipath routing. Presence of parallel paths does not present a complication. The pushback request will travel along both paths and would be merged upstream where the paths meet

[7] The 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.

[8] The server can have some higher level or application-specific attack detection mechanism

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.

After the Pushback Agent determines the limit to request from neighboring upstream routers, it sends a pushback *request* message[9] to those routers. The rate-limit specified in the pushback request is 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). That is, the aggregate will not necessarily receive bandwidth at the upstream router equal to the upper bound; the upstream router is simply requested not to give *more* than the upper bound to the specified aggregate.

We also note that 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. However, since the rate-limiting is also being done at the congested router, more than desired amount of aggregate traffic never goes over the congested link.

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 have to be narrowed, to restrict the rate-limiting to traffic headed for the downstream congested router only. The means that pushback can only be invoked for congestion signatures that include a destination prefix. This is discussed in more detail in [17].

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. 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. Pushback status messages enable the congested router to decide whether to continue rate-limiting (and pushback).

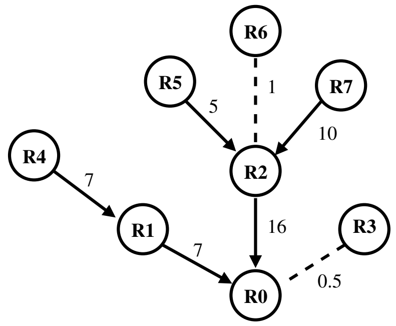

Figure 5: **Pushback status messages reporting the aggregate's arrival rate from upstream.**

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 5, RO 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,

[9] The pushback protocol, including timing and format of messages, is described in [8].

the downstream router uses the status messages to estimate the arrival rate from the aggregate, and then uses the algorithms in Section 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 [20] simulator testing the effect of Local ACC and pushback in a variety of aggregate-based congestion scenarios. 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.

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 6 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.

The results of the simulation are shown in Figure 7. 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 Local 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 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.

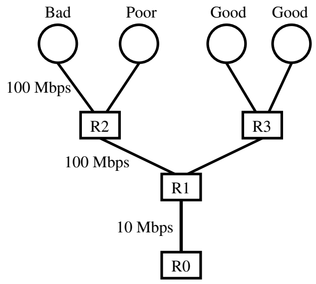

Figure 6: **The topology for a simple simulation.** $R1-R0$ is the congested link.

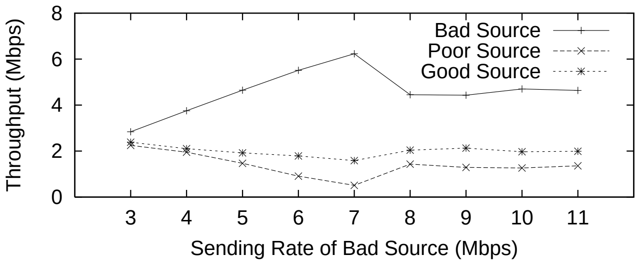

Figure 7: **The effect of pushback in a small topology.**

6.2 DoS Attacks

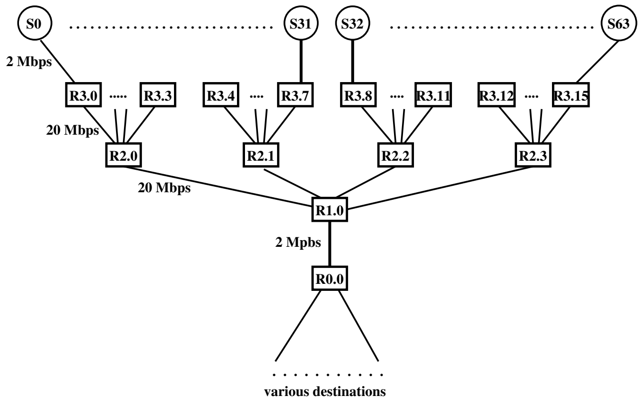

Figure 8: **The topology used in simulations**. $R1.0-R0.0$ is the congested link

The simulations in this section illustrate Local ACC and Pushback with both sparsely-spread and with highly dif-

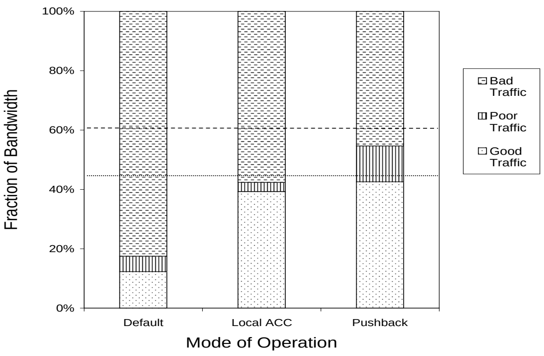

Figure 9: **Bandwidth allocation at the congested link during a sparse DoS attack.**

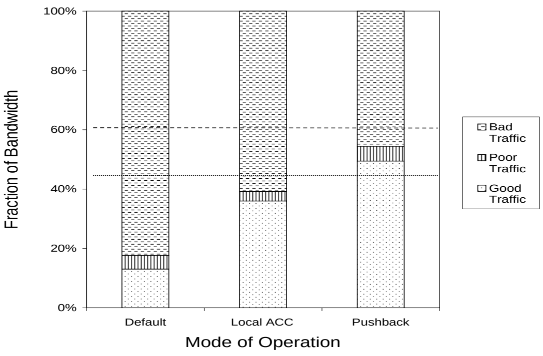

Figure 10: **Bandwidth allocation at the congested link during a diffuse DoS attack.**

fuse DoS attacks. These simulations use a topology 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 have been allocated such that congestion is limited to the access links from the source hosts and to the destination router.

The first simulation scenario, with a sparsely-spread DoS attack, includes four bad sources, four poor sources, and 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 9 shows the results of these simulations. "Default" denotes a simulation with the router not doing any form of ACC. The two lines in the graph denote the quantity of good and poor traffic in the absence of any attack traffic. Both the Local ACC and pushback simulations bring down the bandwidth consumed by the attacker, leading to a significant bandwidth gain for the good traffic. However, as expected, Local ACC 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 DoS 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 10 shows, with diffuse attacks 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 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 11 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 Local ACC 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 De-

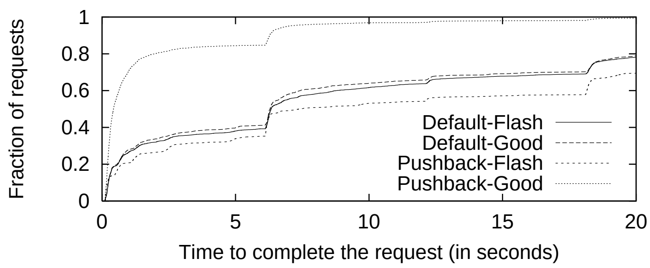

Figure 11: **Time to complete a request during a flash crowd.**

fault 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. The drop rate for the good traffic comes down from 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 a lot of 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 Local ACC 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.

7 Discussion

7.1 Advantages and Limitations of Pushback

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 limitations of adding pushback to ACC.

One advantage of pushback is to prevent scarce upstream bandwidth from being wasted on packets that will be dropped downstream.

When attack traffic can be localized spatially, pushback can effectively concentrate rate-limiting on the malicious traffic within an aggregate. This is very useful when source addresses cannot be trusted because then the congested router cannot narrow the congestion signature by itself.[10] 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 based on the source IP addresses alone (i.e. the attack can be localized spatially but no concise description in terms of source prefixes exists), then pushback is necessary to narrow the attack signature, *even if source addresses are genuine*.

For some DoS attacks, pushback will not be effective in concentrating rate-limiting on the malicious traffic within an aggregate. For example, this would be the case for an attack uniformly distributed across the inbound links. Consider, for example, a reflector attack [21] based on DNS [5]. 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 poor and the bad DNS traffic going to the destination, and will drop from both equally.

Pushback may overcompensate, particularly when it is invoked for non-malicious events such as flash crowds. 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 §4.2). Performing some of the rate-limiting just at the congested router can also help to prevent overcompensation.

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

[10] If source addresses could be trusted, then in some cases the congested router could narrow the attack signature itself, by identifying both the source and the destination address prefixes responsible for the bulk of the traffic in the identified aggregate.

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

7.2.1 Implementation Complexity

The identification of aggregates can be done as a background task, or in a separate machine entirely, 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 at 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 Local 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.

7.2.2 Estimating the Upstream Link's Contribution

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 router. This is simple for point-to-point links; only one router is attached to each interface. However, for routers joined by LANs, VLANs, or frame relay circuits, there are multiple routers attached to an interface. The downstream router in this situation might not be able to distinguish between multiple upstream routers.

One way of dealing with this problem is to send a *dummy* pushback request to all upstream neighbors. The dummy request is similar to the real request, but the recipient does not actually rate-limit the aggregate. The only impact of this request is that the recipient will estimate the arrival rate of the specified aggregate and report it to the downstream router in status messages. These messages help the downstream router to send pushback requests with the appropriate rate-limits to contributing routers.

7.2.3 Incremental Deployment

Pushback can be deployed incrementally by deploying it only on the edges of an *island* of routers, where an island is a set of connected routers. An autonomous system (AS) is a natural island. Assume that the island has N edge routers. When one of these routers gets congested and decides to invoke pushback, it could consider the remaining $N - 1$ edge routers as its upstream links. Using dummy pushback messages it could ascertain the aggregate's arrival rate at each of these routers, and send authenticated pushback request messages accordingly. Thus, even without universal deployment the island can selectively throttle traffic coming in from certain directions.

8 Conclusions

Congestion caused by aggregates differs in some fundamental aspects from that caused by individual flows, and hence requires different protection mechanisms in the network. We have proposed both local and cooperative mechanisms for aggregate-based congestion control to answer this need. While a lot remains to be investigated, simulations carried out till now have shown that they are very promising directions. As part of this work, we are in the process of implementing an experimental testbed using software routers.

Future work will include more thorough investigations of the underlying dynamics and possible pitfalls of these mechanisms. The effectiveness of these mechanisms are deeply tied in with their operating environment. For effective evaluation we need measurement-based answers to questions like "how frequently is sustained congestion caused by aggregates, and not by failures", and "what do attack traffic and topologies look like".

Acknowledgments

The original idea for pushback came from an informal DDoS research group consisting of Steven M. Bellovin, Matt Blaze, Bill Cheswick, Cory Cohen, Jon David, Jim

Duncan, Jim Ellis, Paul Ferguson, John Ioannidis, Marcus Leech, Perry Metzger, Vern Paxson, Robert Stone, Ed Vielmetti, and Wietse Venema. We also thank Randy Bush, Eddie Kohler, Neil Spring and Ed Vielmetti for feedback on an earlier draft of this paper.

References

[1] S. M. Bellovin. ICMP Traceback Messages. Internet Draft: draft-bellovin-itrace-00.txt, March 2000.

[2] J. Borland. Net Video Not Yet Ready for Prime Time. CNET news, February 1999. http://news.cnet.com/news/0-1004-200-338361.html.

[3] 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.

[4] 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.

[5] CERT. Cert incident note in-2000-04, 2000. http://www.cert.org/incident_notes/IN-2000-04.html.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *ACM SIGCOMM*, 1989.

[7] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.

[8] S. Floyd, S. Bellovin, J. Ioannidis, K. Kompella, R. Mahajan, and V. Paxson. Pushback Messages for Controlling Aggregates in the Network. Work in progress. Internet-draft: draft-floyd-pushback-messages-00.txt, July 2001.

[9] S. Floyd and K. Fall. Router mechanisms to support end-to-end congestion control. Technical report, LBL, February 1997.

[10] S. Floyd, K. Fall, and K. Tieu. Estimating Arrival Rates from the RED Packet Drop History, April 1998. http://www.aciri.org/floyd/end2end-paper.html.

[11] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, Vol. 1(4):pp. 397–413, August 1993.

[12] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Transactions on Networking*, Vol. 2(2):pp. 122–136, April 1994.

[13] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, Vol. 3(4):pp. 365–386, August 1995.

[14] L. Garber. Denial-of-Service Attacks Rip the Internet. *IEEE Computer*, vol. 33(4):pp. 12–17, April 2000.

[15] 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.

[16] D. Lin and R. Morris. Dynamics of Random Early Detection. In *ACM SIGCOMM*, 1997.

[17] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High-Bandwidth Aggregates in the Network (Extended Version). http://www.aciri.org/pushback/, July 2001.

[18] R. Mahajan and S. Floyd. Controlling High-Bandwidth Flows at the Congested Router, November 2000. http://www.aciri.org/red-pd/.

[19] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. *USENIX Security Symposium*, August 2001.

[20] NS Web Page: http://www.isi.edu/nsnam.

[21] V. Paxson. An analysis of using reflectors to defeat DoS traceback, August 2000. ftp://ftp.ee.lbl.gov/.vp-reflectors.txt.

[22] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *ACM SIGCOMM*, August 2000.

[23] D. Schnackenberg, K. Djahandari, and D. Sterne. Infrastructure for intrustion detection and response. In *Proceedings of the DARPA Information Survivability Conference and Exposition 2000*, March 2000.

[24] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *ACM SIGCOMM*, 1998.