Mitigating Network Denial-of-Service through Diversity-Based Traffic Management

Ashraf Matrawy, P.C. van Oorschot, and Anil Somayaji

Carleton University, Ottawa, ON K1S 5B6, Canada,
amatrawy@sce.carleton.ca, {paulv,soma}@scs.carleton.ca

Abstract. In this paper we explore the feasibility of mitigating network denial-of-service (NDoS) attacks (attacks that consume network bandwidth) by dynamically regulating learned classes of network traffic. Our classification technique clusters packets based on the similarity of their contents—both headers and payloads—using a variation of n-grams which we call (p, n)-grams. We then allocate shares of bandwidth to each of these clusters using an adaptive traffic management technique. Our design intent is that excessive bandwidth consumers (e.g. UDP worms, flash crowds) are segregated so that they cannot consume bandwidth to the exclusion of other network traffic. Because this strategy, under congestion conditions, increases the packet drop rate experienced by sets of similar flows and thus reduces the relative drop rate of other, dissimilar flows, we characterize this strategy as *diversity-based traffic management.* We explain the approach at a high level and report on preliminary results that indicate that network traffic can be quickly and concisely learned, and that this classification can be used to regulate the bandwidth allocated to both constant packet and polymorphic flash UDP worms.

Keywords: network denial of service, flash worms, traffic shaping, network security, diversity

1 Introduction

In recent years the stability and usability of the Internet has been challenged by numerous uses and abuses unforeseen by its original designers. Peer-to-peer applications saturate links with searches and file transfers. Web servers and small service providers are overwhelmed by flash crowds initiated by the rapid spread of ideas and links on the "blogosphere" while email servers are flooded with vast quantities of unsolicited commercial email. Most disturbingly, self-replicating autonomous programs (worms and viruses) flood network connections through scans and infection attempts, some spreading worldwide in seconds [1].

Many researchers have chosen to address each of these issues as isolated problems; we believe, however, that progress can also be made by recognizing what these problems have in common. In all of these situations, the actions of a few applications can consume all available bandwidth and in so doing prevent other hosts, applications, and users from communicating. While the denial-of-service (DoS) problem in general has received much attention in recent years, this type of bandwidth-consuming network DoS (NDoS) has received less study. NDoS is the one problem, however, to which no host or network is immune—no matter how well it is otherwise protected. Fundamentally, bandwidth forms a commons, in that it is a resource used by all but is completely controlled by nobody. Although it is possible to create mechanisms to allocate the bandwidth commons, such allocations will impose some limitations on communications—the primary purpose of the Internet. The question we pursue is that of how bandwidth may be allocated in a way that better reflects the differing needs of Internet users and applications. In doing so we try to prevent the actions of a few categories of excessive bandwidth consumers from disrupting the activities of other, more moderate consumers.

Rather than attempting to specifically identify undesirable sets of packets (whether they be malicious or simply less important by some measure), our approach is to allocate bandwidth on the basis of packet similarity. In some circumstances these commonalities may be shared destination ports or host IP addresses; in others, it may be recurring payload substrings. In our approach, packets sharing relatively high-frequency patterns (in header and/or payload) are identified as a set, and each of these sets is limited to a controlled fraction of network bandwidth. So long as such a set is "adequately represented" in the network flow, its bandwidth limit does not shrink—nor grow. Thus, while such sets are guaranteed representation in the outgoing packet stream, in the case of bandwidth starvation they must also share bandwidth with other identified packet sets. Because this strategy tends to increase the diversity of packets within a given network connection (under saturation conditions), we refer to it as *diversity-based network traffic management.*

To study the feasibility of this bandwidth management strategy, our research has focused on a simple pattern schema that we call (p, n)-grams. Like the better-known n-grams (cf. Section 2.2), (p, n)-grams are fixed-length strings of byte-length n; unlike n-grams, they are at a fixed offset p within a packet—thus allowing for very efficient pattern matching, even within payloads. More specifically, our work includes: (1) studying the patterns of (p, n)-grams present in captured network traffic; (2) creating and analyzing high-speed online algorithms for extracting sets of (p, n)-grams suitable for dividing packets into similarity sets; and (3) developing an architecture for dynamically allocating bandwidth between sets of similar packets.

Our Contributions. In this paper we propose that network denial-of-service attacks can be mitigated by adaptively clustering network packets and by allocating bandwidth on the basis of such clusters. To implement this strategy, we propose a new measure of packet similarity, namely (p, n)-grams, and a simple feedback algorithm for learning sets of (p, n)-grams that can be used to subdivide network traffic. We also propose an architecture for using this algorithm to regulate network traffic so as to minimize the impact of large aggregates of similar packets in congested conditions. We then present experimental data that indicates that the proposed algorithm can quickly and concisely learn the patterns of normal network traffic. With the injection of simulated worm traffic into live captured network data, we also show that the proposed mechanisms can limit the bandwidth allocated to both constant and random-payload UDP flash worms automatically and autonomously.

Outline. In what follows, Section 2 presents related work. Section 3 discusses the rationale for diversity-based network traffic management and our architecture for managing network traffic. Section 4 describes our approach to (p, n)-gram packet analysis. Section 5 presents the results of preliminary experiments. Concluding remarks in Section 6 include a discussion of ongoing challenges and plans for future work.

2 Related Work

Although it has been generally realized that network traffic must be managed to prevent communication disruptions, no central strategy has emerged that is effective at dealing with all aspects of the problem. Instead, researchers have developed many different mechanisms, each designed to address certain circumstances. To organize our review of past work, we divide these mechanisms into two classes: those that address the congestion control problem by managing overly aggressive bandwidth consumers, and those that identify and respond to malicious network traffic.

2.1 Congestion control for aggregates of flows

Congestion control has been a basic design principle of the Internet from its earliest days. Indeed, the robustness of the Internet can be attributed in part to the end-to-end congestion control mechanisms built-in to TCP [2]. End-to-end congestion control, however, is based on the assumption that implementors of network protocols and applications are willing to cooperate to maximize network efficiency and fairness. While such assumptions were once well-founded, this is no longer the case.

Perhaps the first signs of trouble were in the form of UDP-based media applications. Early versions of these systems were not good network citizens, in that they would not throttle their communications in response to developing congestion. Therefore, they would gain an unfair share of bandwidth when competing with TCP streams that would honor such implicit congestion messages. The situation was even worse for multimedia multicast applications. These issues motivated a new area of research and development: unicast [3] and multicast [4] TCP-friendly protocols and applications. While the performance of these TCP-friendly mechanisms is satisfactory in some cases, the growing need for better congestion management has resulted in the development of mechanisms such as Explicit Congestion Notification [5] and Random Early Detection Gateways [6] where the network provides some congestion control support to end systems.

The main characteristic of the above architectures and mechanisms is that they are targeted at misbehaving *individual* flows. A flow is normally defined as a set of IP packets that are exchanged on fixed TCP or UDP ports between two

IP addresses. As noted by Estan and Varghese [7], a large fraction of network bandwidth is sometimes consumed by a few large flows (e.g. by large file transfers, multimedia streams, etc.). In such situations, a sampling strategy can be used to find these "heavy hitters"; in the case of flash crowds and distributed network denial-of-service attacks, though, there are no such heavy hitters to identify. To address this limitation, network analysis systems such as AutoFocus [8] cluster flow state descriptors in order to discover sets of shared features in the high-dimensional space defined by IP addresses, protocols, and ports.

Rather than focus on network analysis, others have developed tools with which to manage network bandwidth. Systems such as Diffserv [9, 10] label and prioritize traffic according to pre-negotiated quality of service classes. Traffic shaping mechanisms [11, 12] can be used to limit the bandwidth allocated to specific quality-of-service classes, hosts, and/or ports. Existing traffic shaping systems are very good at managing traffic according to pre-established policies; however, static rules are not sufficient to manage the transient problems created by flash crowds, worms, or NDoS attacks.

Other researchers have recognized these limitations and have sought solutions. In particular, Mahajan et al. [13] (see also [14]) proposed that DDoS attacks be dealt with as an *aggregate congestion control* (ACC) problem. They combined a local mechanism (at the routers) to detect signatures of flow aggregates that are causing congestion, and a co-operative mechanism—called *pushback*—to notify other routers of these signatures so that they may take action against these aggregates to limit their impact. The local mechanism uses the destination address to detect high-bandwidth aggregates, which are represented using *congestion signatures*; these signatures are passed to other routers by the pushback mechanism when it is deemed necessary to communicate a deteriorating congestion status at a certain router. The intent is to stop the spread of high-bandwidth aggregates as close as possible to their source. As originally conceived, pushback would seem to require routers to store large amounts of information on flow state; work such as that by Yaar et al. [15], however, shows that this need not be the case if end points explicitly manage their bandwidth commitments.

While pushback has the potential to stop certain classes of NDoS, it also has some fundamental limitations. First, pushback requires that routers coordinate their responses to misbehaving flows. A large number of routers would need to adopt the pushback mechanism for it to be effective; such adoption may not be wise, however, since if the pushback communications channel is compromised, pushback itself could become a very effective tool for NDoS. A more fundamental issue, though, is that there may not be any identifiable set of flows (at least at the level of IP addresses and ports) that are responsible for observed congestion. Such a situation may arise if the congestion comes from a flash crowd or a rapidly propagating worm.

2.2 Network-Level Anomaly Detection

Another perspective for addressing the DoS problem arises from the observation that malicious network traffic typically has different structure and content from non-malicious traffic. If all non-malicious traffic can be profiled as "normal," then any observed abnormal traffic can be classified as malicious. While there are many methods for detecting security violations at the network level, anomaly detection is arguably the approach that is most capable of detecting attacks that exploit previously-unseen vulnerabilities. Because malware developers discover new vulnerabilities on almost a daily basis, and because attacks by worms and viruses are both fast and automated, there is a pressing need for methods that can both detect and respond automatically to such "zero-day" threats.

One of the earliest methods for detecting anomalous network traffic was Heberlein et al.'s Network Security Monitor (NSM) [16]. Their system used a centralized monitoring host to record the 5-tuples associated with normal network flows. After a training period, any new 5-tuples were classified as an anomaly. In 1999, Hofmeyr developed LISYS [17], an intrusion detection system similar to NSM except that its architecture permitted the set of normal network flows to be distributed across a set of hosts. While this model of intrusion detection is useful in a relatively static network environment, the rapid evolution of new services, protocols, and servers on today's Internet means that the presence of a novel 5-tuple connection is not sufficient to indicate malicious traffic.

More recently several approaches to automatically characterize and respond to rapidly propagating worms have been proposed. Among the most promising is Singh et al.'s *EarlyBird* system [18, 19], which relies on two basic observations associated with fast worm propagation: (1) most such worms produce a substantial volume of traffic containing identical payload substrings; and (2) these similar packets are relayed between an increasing number of distinct source and destination addresses. With these heuristics and a combination of several highly efficient, scalable algorithms such as Rabin fingerprints [20], EarlyBird is able to automatically identify packets with common substrings and worm-like propagation patterns. Once a worm has been identified, the observed payload substrings are then used to block subsequent worm packets. In online experiments over the course of several months, their system is reported to have detected many previously known worms along with new, previously unseen worms, all with zero false positives. However, in order to avoid false positives, they had to create a packet "white list" that included such patterns as the protocol identification strings of HTTP and SMTP and the repeated packets corresponding to some BitTorrent-based P2P file sharing.

While EarlyBird is (to our knowledge) the first to report a working online implementation, similar systems are in development. Kreibick and Crowcroft's [21] *Honeycomb* system also detects worms by extracting high-frequency substrings within packets; a novel aspect is to extract common substrings only in *honeypots* to increase the likelihood of finding malicious traffic. A similar system also based on packet content inspection, *Autograph* [22], uses a simple port-scan-based flow classifier to reduce the amount of traffic on which signature extraction is applied.

In another approach to network intrusion detection, Wang and Stolfo [23] propose a system which builds byte distributions (taking into consideration port number and packet length) of "normal traffic". Incoming packets are compared with these distributions to measure their similarity with normal traffic. Packet similarity is measured by computing the Mahalanobis distance [24] between distributions.

While these systems use packet classification methods ranging from simple heuristics to complex statistical models, what they have in common is that they classify network traffic into two groups: legitimate and illegitimate traffic. The difference between these two classes, however, is not always obvious even to a human observer. For example, "legitimate" flash crowds can be created maliciously, e.g. by posting a targeted web server's home page to a popular weblog. This ambiguity has been one of the prime motivators for our research.

3 Diversity-Based Traffic Management

While many of the known approaches to managing network traffic have appealing characteristics, we believe current approaches will fail to address the threats of tomorrow, especially as attackers adapt and attempt to subvert deployed anomaly detection systems—especially those targeted at flash worms [25] (see [26]). One way to avoid the trap of attacker innovation is to simply re-frame the problem by dividing it in two. Proceeding in this direction, we separate the problem of protecting vulnerable hosts from that of managing network bandwidth. For host protection, many mechanisms are at our disposal: virus scanners, automated code patches, buffer overflow defenses, code diversity, and even host-based (personal) firewalls and intrusion prevention systems. Some combination of these and future technologies should be adequate to provide a reasonable level of practical protection for any host that a user chooses to secure. To protect the network from less well-maintained systems, though, we need mechanisms that prevent any collection of hosts or services from consuming an excessive share of bandwidth.

General Approach. With this framework, security threats such as flash worms and distributed denial-of-service attacks simply become additional types of overly aggressive applications, ones that can potentially be managed in the same way as flash crowds, spam, and peer-to-peer file sharing systems. An initial thought might be not trying to determine "good" versus "bad," but rather "disruptive" versus "non-disruptive" traffic. However, this change in perspective does not necessarily make the problem easier. By our definition, *disruptive network traffic* is that which prevents the transmission of other packets due to congestion. Because disruptive traffic can only be recognized in relation to other traffic flows, it is defined by context, not content. Moreover, that context implies a value judgment that some are the "disruptors" while others are "being disrupted." As with the problem of identifying worm traffic, it can be dangerous for an automated system to make value judgments; it can be too easy for a computer to make a mistake.

Thus, what we really want is a technique for managing disruptive network traffic without ever having to explicitly identify what is disruptive. One general way to accomplish this goal is to classify packets (as explained below) into multiple sets and then apportion bandwidth between these sets. If attackers do not know the (current) basis upon which packet sets are defined, or if they cannot create packets belonging to arbitrary sets, then attackers cannot consume all available bandwidth; instead, they can only consume the bandwidth fraction that is assigned to the sets in which their packets fall.

Rather than using an explicit, static tag (as in Diffserv [9, 10]), we propose to classify packets using their "intrinsic" properties. We assume that attackers can create and inject arbitrary packets into network connections, and thus we cannot consistently use features such as source IP addresses to sort packets—otherwise an attacker could consume an arbitrary fraction of bandwidth by injecting packets with those same features. Instead, we dynamically choose classification patterns from those present in observed traffic. If we assume that an attacker cannot observe or accurately model most traffic at a given Internet routing nexus, then the hope is that such a bandwidth allocation scheme cannot be directly subverted by an attacker.[1] Our research, then, has been focused on methods for grouping packets on the basis of features present in actual network traffic, with the goal of using these groupings to manage bandwidth allocation.

Since we allocate bandwidth on the basis of packet features, our strategy is designed to give small sets of packets with unusual shared features a larger share of bandwidth than they would normally be allocated under link saturation. Because outgoing packets will have a higher frequency of such unusual patterns than they would otherwise (when congested), we characterize our method as increasing the diversity of packets. Thus, we avoid the problems of distinguishing between disruptive and non-disruptive traffic by favoring increased diversity of network flows.

Architecture. Recall that we are not differentiating between legitimate and illegitimate traffic; we cannot simply drop certain kinds of packets. Rather, we characterize packets (see above, and below) and then forward them based upon their class membership. To do this, we propose to use traffic shaping mechanisms to differentially queue and forward packets. Traffic shapers [11, 12] are typically used to allocate bandwidth between ISP customers or to limit commonly used applications that are heavy bandwidth users (e.g. multimedia streaming, peer-to-peer file sharing)—in other words, they are used to regulate traffic based upon port and IP address patterns. We propose to augment such traffic shaping mechanisms with more flexible and adaptive methods for classifying packets.

To be more specific, consider the traffic shaping architecture in Figure 1. In the context of a simple LAN connected to the Internet, our traffic shaping mechanism would be implemented on the LAN's local router (to manage outgoing packets) and on the Internet Service Provider's router (to manage packets inbound to the LAN). As packets arrive, they are placed by the classifier into one of a number of packet queues associated with an outbound interface. Packets from these queues are chosen for forwarding in a round-robin fashion. Periodically the classification rules are updated by a traffic analysis module that runs in parallel and works on periodically sampled packets. While this basic archi-

[1] As suggested by the wording, we have yet to prove this.

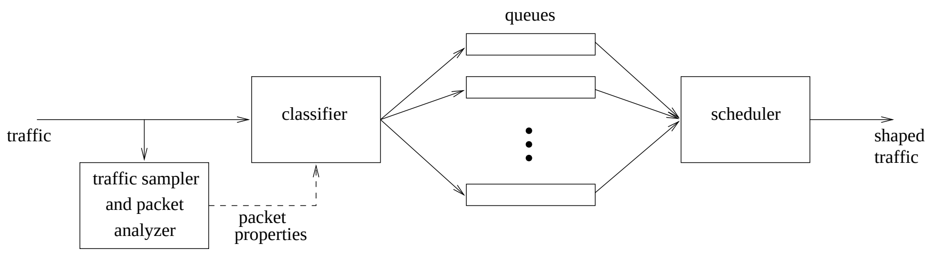

Fig. 1. Typical architecture for an adaptive traffic-shaping system.

tecture implies that all packets are to be forwarded on a best-effort basis, we can also support multiple quality-of-service bands by replicating this system for each band; similarly, more complex topologies can be accommodated by placing a shaping module in front of any outbound interface within a given router.

Given a sufficiently efficient packet classification mechanism, this architecture can be implemented both at core routers and edge routers. In either case, it will help the router manage network bandwidth in the face of a surge in disruptive traffic. Currently this architecture is more appropriate for bandwidth-limited edge routers rather than the (apparently) over-provisioned core of the Internet. However, the introduction of new bandwidth-demanding applications may increase the utility of bandwidth management schemes such as ours in core routers.

Given this simple architecture, we next need a packet classification method that can group packets quickly (at router speeds) but in a way that cannot be easily predicted by an attacker. Although our classification may be approximate—it need not divide packets into any human-recognizable categories—our chances of limiting bandwidth allocation to disruptive traffic will be higher if packets within a given queue are somehow related (e.g. worm packets ideally should be classified within a set distinct from non-worm traffic, and to limit the worm bandwidth allocation to that allocated to one queue, or at worst a small number of queues). Thus, we have focused on developing an online method for finding common features between packets—features that can be used to assign packets to queues. The method we chose to develop is based on a simple scheme, which we call (p, n)-grams.

4 Packet Classification using (p, n)-grams

We now describe how we use a variation of n-grams to classify traffic, and explain our variation of traffic shaping using a set of queues managed adaptively, based on the continuous analysis of incoming traffic.

4.1 The introduction of position p into n-grams

As described earlier, a (p, n)-gram is a byte string of length n located at an offset p in a packet within a data stream. Other systems (see Section 2.2) classify packets using the presence of substrings or the the relative distribution of 1-grams; in contrast, we combine substrings and offset positions in our representation for two reasons.

First, position matters in the context of an IP packet, especially within IP, TCP, UDP, and application-level protocol headers. For example, a four-byte IP address within a packet can have many different meanings depending upon its position within the packet: it can represent an IP source address, an IP destination address, a HTTP referrer host, or even a distant SMTP relay. Thus, by including position we have a method for capturing the important semantic features of IP packets.

Our second motivation for using an offset is speed for the online classification stage. To check for the presence of a (p, n)-gram in our system, one needs to perform a position computation (to find the offset p within the packet) and then a comparison of n bytes. (Note that such an offset/substring comparison is what a router does when identifying the destination address of an IP packet.) In contrast, typical position-independent n-gram techniques require a comparison with every byte within a packet. Techniques such as Rabin fingerprints [20] can significantly reduce the cost of multiple n-gram comparisons; nonetheless, (p, n)-gram comparisons are simple and efficient enough to scale to high speed links.

4.2 Adaptive learning and classification

While it is necessary to have a simple pattern schema (such as (p, n)-grams) with which to classify incoming packets, such a schema is not sufficient for our purposes. Because we have no pre-specified notion of how packets should be classified, we also need a learning algorithm that can discover appropriate classes autonomously. While this learning algorithm need not run at wire speeds, it must be fast enough that it can update the set of patterns frequently (e.g. every few minutes). While there are numerous classification and clustering algorithms in the literature, for performance reasons we have focused on a simple feedback learning strategy based on candidate (p, n)-grams extracted from samples of incoming packets. When the existing set of classification (p, n)-grams are not matching incoming traffic with sufficient fidelity, appropriate candidate (p, n)-grams are chosen as supplements and replacements.

More specifically, our system iteratively builds up a set of (p, n)-grams that allow network traffic to be partitioned into multiple “equivalence classes”. The (p, n)-grams selected are initially those having the highest frequency in recent traffic; by merging sets of (p, n)-grams describing small aggregates and splitting ones describing large aggregates, though, this method attempts to converge on a description of “normal network traffic”. When bandwidth becomes scarce, this iterative process is halted and the learned (p, n)-gram sets are used to (implicitly) decide which packets get dropped, i.e. the unserviced packets in long queues.

For example, assume we have a router with interfaces A and B, and we wish to shape traffic coming in from A and out to B. To do this shaping, we establish a fixed number q of queues Q associated with the outgoing interface B. When a packet is received on A that is destined for B, it is placed in one of the queues. Packets from the queues are selected in a round-robin[2] fashion and are sent out interface B.

Associated with each queue $Q_i \in Q$ is a bounded set of (p, n)-grams. Note that the position p in a (p, n)-gram may specify anywhere in an IP packet, including in protocol headers and application-level payloads. To choose the queue in which to place a packet s, we first see whether any of the (p, n)-grams associated with Q_0 is present in s; if so, then s is placed in queue Q_0. Otherwise, test the next queue in the same fashion. Queue Q_{q-1} (called the *default queue*) is special, as it has no (p, n)-grams associated with it; if s is not placed into any of the other queues, it is placed in Q_{q-1}.

Initially, each queue is associated with a recently observed frequent (p, n)-gram. At fixed time intervals (e.g. every five minutes), the performance of the selected (p, n)-grams is examined and the system determines whether too many packets are falling into the default queue, and whether the distribution of packets between the queues is too imbalanced. If the currently selected (p, n)-grams are performing poorly according to this criteria, and if we have not dropped any packets over another fixed time interval (e.g. one hour), we then re-balance the queues (with respect to future traffic) by updating the sets of (p, n)-grams associated with each poorly-performing queue. Alternately, if we have dropped packets by exceeding the maximum length of any queue, then we *do not* rebalance the queues.

The sets of queue-associated (p, n)-gram sets are updated in three ways. First, small queues are enlarged through merging: the (p, n)-gram sets associated two or more small queues are merged to form one new queue. Second, to reduce the number of (p, n)-grams in the system, when necessary the least frequently used (p, n)-grams are removed from the system. The third way (p, n)-gram sets are updated is through *splitting*: large queues are split through the creation of a new (p, n)-gram set that matches a subset of the large queue's traffic. To this end, we maintain a buffer of recent packets (e.g. the past 1000) that were placed within each queue. To split a queue, the (p, n)-grams contained within the queue's buffer are extracted and sorted by frequency. One or more of these (p, n)-grams are then selected such that they match a significant fraction, but not all, of the packets within the queue's buffer (e.g. (p, n)-grams that together match more than 20% but less than 50%). These new (p, n)-grams are then used to create a new queue. Splitting and merging are depicted in Figure 2.

As should be apparent, there are a number of parameters and implementation details that are not specified in the above description. Indeed, here we have defined an entire class of learning algorithms. Section 5 describes preliminary re-

[2] Currently we use round-robin to allocate an equal share of bandwidth to every queue. Later we plan to study the effect of the service mechanism and the effect of the drop policy (currently we are using simple drop-tail).

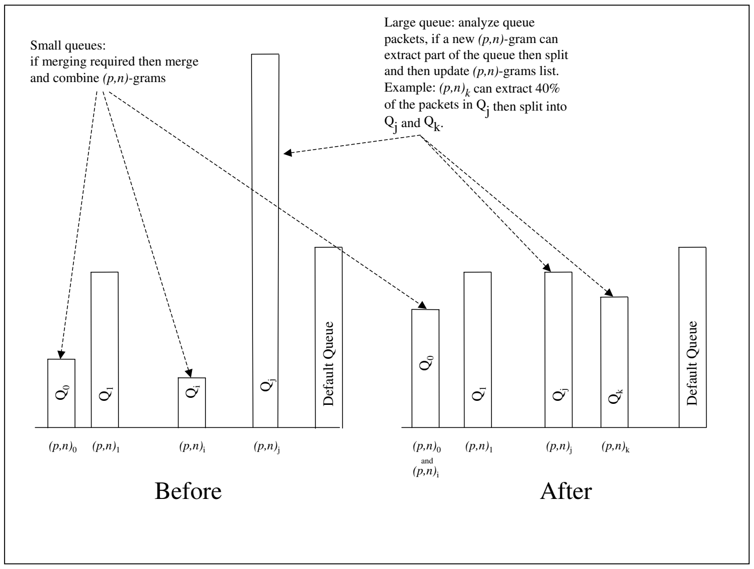
Small queues:
if merging required then merge
and combine (p,n)-grams
Large queue: analyze queue
packets, if a new (p,n)-gram can
extract part of the queue then split
and then update (p,n)-grams list.
Example: $(p,n)_k$ can extract 40%
of the packets in Q_j then split into
Q_j and Q_k.
Q_0
Q_1
Q_i
Q_j
Default Queue
$(p,n)_0$
$(p,n)_1$
$(p,n)_i$
$(p,n)_j$
Before
Q_0
Q_1
Q_j
Q_k
Default Queue
$(p,n)_0$ and $(p,n)_i$
$(p,n)_1$
$(p,n)_j$
$(p,n)_k$
After

Fig. 2. The adaptive split/merge operation of queues

sults obtained with one instance of this class. While the tested instance performs well in many respects, ongoing work is focused on studying other variants.

4.3 Malicious encrypted and polymorphic traffic

The use of position in (p, n)-grams introduces the specific drawback that a small change in a string's position will prevent a match. This problem would be very significant if we were concerned about adversaries attempting to evade our packet classification system using encrypted or polymorphic traffic. However, so long as there are sufficient number of high-frequency (p, n)-grams present in other packets, we believe such behavior will actually hurt, not help an adversary, as now explained.

Packets with random contents (both header and payload) are highly unlikely to match any of the (p, n)-grams associated with queues; thus, streams with encrypted payloads and no repeated header patterns—e.g. a polymorphic flash worm—will be assigned to the default queue. At the same time, we expect that by construction normal traffic (future packets) will match many of the selected (p, n)-grams, and thus will be allocated most of the router's outgoing bandwidth.

An attacker wishing to consume a maximal amount of bandwidth may pursue a strategy of trying to construct packets that match at least one (p, n)-gram assigned to each queue—perhaps by adaptively constructing packets or by somehow causing the classification algorithm to replace (p, n)-grams matching normal traffic with ones matching malicious traffic. So long as selected (p, n)-grams are dependent on the specifics of local network traffic to which the attacker has no direct access, we believe it will be very difficult for an attacker to predict the constantly-changing set of (p, n)-grams corresponding to normal traffic. It may be possible for an attacker, however, to gradually train the system to increase his bandwidth allocation; we plan to study such adaptation (and appropriate algorithmic countermeasures) in future work.

5 Preliminary Experiments

To date we have focused on using offline experiments to evaluate the feasibility of (p, n)-grams for managing network traffic. In particular, we have focused on three central questions. First, are there a sufficient number of high-frequency (p, n)-grams (both in headers and in payloads) to enable the partitioning of network traffic? Second, does there exist a feedback learning algorithm that is able to learn a relatively small set of (p, n)-grams with which to partition network traffic? And finally, would this algorithm properly manage flash worm-like activity?

Run#	Capture Date	Packets	% UDP	% TCP	% Other
1	Aug. 16, 2004	3437573	52%	16%	31%
2	Aug. 17, 2004	2156483	83%	16%	2%
3	Aug. 18, 2004	1283486	25%	71%	3%
4	Aug. 19, 2004	1006355	33%	63%	4%

Table 1. Details of selected data sets analyzed.

To address these questions, we have studied the distribution of (p, n)-grams in packets captured on the link between the outside world and one of our small labs of seven Linux workstations, three Windows XP workstations, and four Linux servers. (Packets travelling from our network to the Internet were not analyzed, but only downstream ones.) For this paper, we report analysis of four days of captured traffic, the details of which are listed in Table 1; graphs presented, however, are based on the traffic of August 17, 2004. Data from other days shows a similar level of variability. Because this lab is essentially self-contained and is used for common tasks such as email (using an independent email server within the lab) and web surfing, traffic on our uplink, while not generally representative of Internet traffic, nevertheless has proven to be a rich data source that has helped us test and refine our ideas.

To understand the feasibility of using (p, n)-grams to classify network traffic, we first studied the distribution of (p, n)-grams. This distribution is best characterized as a "heavy-tail" distribution, with a few members having very high frequencies but most having relatively low frequencies (see Figure 3). We have found that high frequency (p, n)-grams typically match structural features

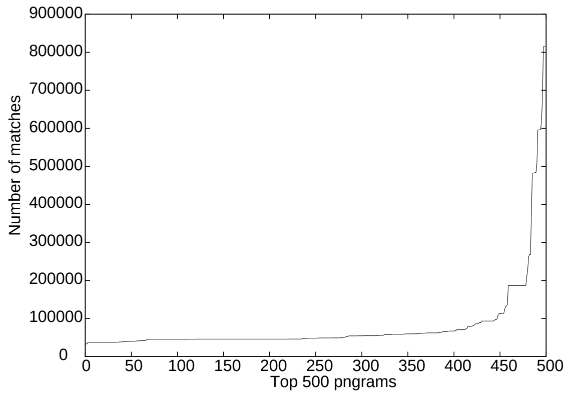

Fig. 3. Frequency graph of the 500 most frequent (p, n)-grams, with p allowed to vary over the entire packet length. Graph is based on the August 17, 2004 dataset with $n = 4$.

of packets such as the IPv4 version number and padding fields. Such structural patterns are the exception, though—most (p, n)-grams present in a fixed data set will be located in the payload (assuming payloads are on average significantly longer than the IP headers), and most of these (p, n)-grams will match the payloads of a small number (perhaps no more than one) of these packets.

While such analysis told us something about the data, it could not directly tell us whether traffic could be effectively classified using (p, n)-grams. So, we then proceeded to implement a version of our feedback learning algorithm suitable for simulating queuing behavior using an offline data set. Our implementation reads in packet capture files in `libpcap` format, one file at a time, classifies these packets into (p, n)-gram queues, and rebalances the queues before reading the next file. Since our captures are stored in 10-minute increments, this means that rebalancing happens at the same time intervals.

The classification program fixes $n = 4$ and starts with 50 queues, with the 1000 frequent (p, n)-grams (from an earlier day) initially assigned to these queues (most frequent (p, n)-gram to least, highest priority queue to lowest, assigned in a round-robin fashion one (p, n)-gram at a time). Traffic is then classified in 10-minute intervals, with (p, n)-gram frequency statistics being preserved for the most recent 3 hours of traffic. After processing a file, all queues are examined. If a queue received more than 2% of recently observed packets (i.e. within the past 3 hours), it is split. Similarly, queues are merged with the nearest lower priority queue if they each have received less than 0.25% of recently observed packets. The default queue is never merged with any other queue; however, it is split if the 2% threshold is exceeded. To prevent the accumulation of non-matching (p, n)-grams, ones that have not matched any packets for the last 3 hours are deleted.

When a queue is split, the program looks for a (p, n)-gram present in the last 10 minutes of packets placed in the queue (up to 1000 packets) that matches at least 20% of these packets but no more than 50%. (The search proceeds from the most frequent to the least frequent (p, n)-gram.) If no such (p, n)-gram is found, the queue is not split during the current time slice (but it may be split on the next time slice). When the new queue is created, we do not have a history of its behavior; thus, we allow it to exist for three hours (our history window) before allowing it to be merged or deleted.

Note that in this implementation there is not a fixed number of queues, and further there is not an upper bound on the total number of queues that may be created; however, in practice the merge/split bounds regulate the number of queues, as will be seen.

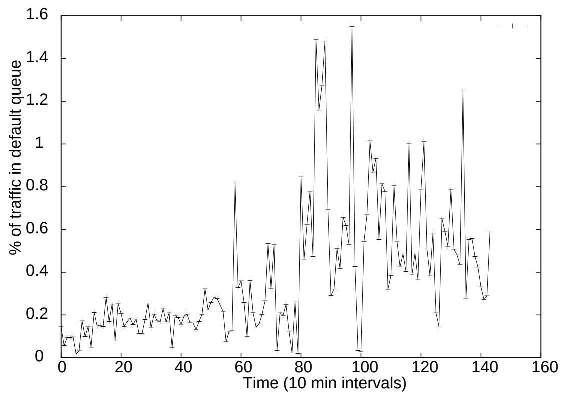

Fig. 4. Percentage of packets placed in the default queue. Despite the variation, note that the default queue is almost always below the target of 2% (the maximum value on the vertical axis). Graph is based on the August 17, 2004 dataset with $n = 4$.

We found that this feedback algorithm was able to successfully subdivide traffic on a regular basis. As indicated in Figure 4, the default queue size remained consistently small. Occasionally surges of traffic would appear within it; if they persisted or were large, then an attempt was made to split the default queue to bring it back down to a reasonable size. We were concerned that the system might accumulate an excessive number of low-traffic queues; even with the 3-hour minimum queue lifetime, however, the system used between 100 and 200 queues at any given time, with approximately 10 (p, n)-grams per queue on most time intervals. Thus, the feedback algorithm was able to consistently learn a relatively small set of (p, n)-grams with which to describe recent network traffic.

Note that the above results are for $n = 4$. By reducing the size n of matching substrings, we should be able to reduce the number of (p, n)-grams necessary to describe normal network traffic. We plan to study alternative learning algorthims that automatically adjusts the size of n so as to minimize the number of (p, n)-grams needed to divide network traffic into a fixed number of queues.

Simulated UDP worm. To examine how the system might respond to a surge of malicious traffic, we simulated the activity of a constant payload UDP worm (such as SQL Slammer [27]) and a random payload (pseudo-polymorphic) UDP worm. For each case, we generated UDP packets using the `nemesis` packet generation tool using random source ports, source IP addresses, and destination IP addresses. The constant payload worm had a fixed (random string) payload of 714 bytes and a constant destination port of 1434. The random payload worm had a payload size between 357 and 1434 bytes and a random destination port.

As we expected (and hoped), both simulated worms initially saturate the default queue. On the first split operation, the constant-payload worm is placed in its own queue. On subsequent iterations (over the next 30–60 minutes), with no simulated packet dropping, every other non-default queue is merged into a second queue (pairwise merging happens on every time step). This merging happens because the large number of "worm" packets distort the overall packet statistics. By instead simulating dropped packets from overflowing queues, though, the program turns off rebalancing and keeps the (p, n)-gram assignments from before the attack, and the worm stays in the default queue.

The random payload worm behaves much the same; however, because it has no high-frequency (p, n)-grams, even without finite queues its packets remain in the default queue. Like the constant-payload worm, though, without finite queues the large number of worm packets bias the queue statistics and cause all remaining normal traffic to be placed into one queue after several iterations of the algorithm.

These results suggest that (p, n)-grams can be used to regulate network traffic, at least in the case of traffic destined for a small network. Further, this regulation appears to be effective at limiting the bandwidth that would be allocated to simple flash worms, even if they were polymorphic. While these results do not necessarily generalize to the behavior of larger, more complex networks, they are promising enough to warrant further experimentation and research.

6 Discussion and Concluding Remarks

We start this discussion by summarizing the methodology we are using in this approach. We have proposed to address the NDoS problem by adaptively learning sets of network traffic and then apportioning network bandwidth between these sets. We have also hypothesized that network traffic can be flexibly subdivided through the (p, n)-gram measure of packet similarity. Even though our framework and algorithm do not distiguish between malicious and non-malicious network traffic, nonetheless they appear to be able to mitigate the effects of some kinds of flash worms. We also hypothesize that the same mechanisms will be able to moderate the bandwidth consumption of other disruptive traffic such as flash crowds and high-volume spam.

Given the results of the previous section, we believe that there is much potential in our approach to managing network traffic. However, as itemized below, a number of important issues require further attention.

a) *The nature of (p, n)-gram learning:* For diversity-based traffic management to work against a hostile adversary, it is essential for normal traffic to be partitioned into groupings that are non-random yet cannot be easily predicted or overly affected by an attacker. If the partitioning is random, then malicious traffic will be placed into many different queues, thereby negating the advantage using separate queues; if the partitioning is too predictable, then an adversary may be able to increase his bandwidth allocation by constructing packets that will match many queues; if the partitioning can be too easily influenced, then the attacker can change it to suit his needs. The input dependence and non-linear feedback of our current algorithm seems to preclude simple analysis and would seem to require significant effort for an attacker to overly influence; nevertheless, we do not have enough data to ascertain how our algorithm will behave in practice. We are encouraged, though, that our offline experiments indicate that this algorithm tends to find fuzzy groupings that vary over time.

b) *The impact of packet re-ordering:* Since TCP and UDP flows are sometimes split across multiple queues, it is likely that packets within such flows will be re-ordered and differentially delayed. We have not specifically studied this issue, but we do not think that such perturbations will significantly impact end-to-end performance in non-congested situations. If it does turn out to be a concern, then such reordering can be eliminated by using the queue structure only to determine which packets should be dropped (with no congestion, packets would be relayed in the order received).

c) *Randomized payloads:* If a polymorphic worm has no repeated pattern of inter-packet (p, n)-grams, then our classifier will place most worm packets into the default queue. Since other encrypted traffic, such as the payloads in SSH and SSL, will also lack such structure, the worm may end up overwhelming such legitimate connections. One way to minimize this problem is to partition encrypted packets by header (p, n)-grams which match on IP addresses and ports. It remains to be determined whether special code will be needed to do such partitioning, or whether such patterns will be learned automatically.

d) *The composition and distribution of (p, n)-grams in more complex environments over longer time periods:* Are there a sufficient number of high-frequency (p, n)-grams such that the traffic of an enterprise or a university can be effectively partitioned? If we have sufficient (p, n)-grams, then is it also feasible for an attacker to predict which (p, n)-grams we will choose? It is possible that the greater sample sizes of larger networks will cause our method to always find similar sets of (p, n)-grams across different networks. If this turns out to be the case, then we may need new strategies for core Internet routers.

e) *Conflict with existing network policies:* One capability that we may need to provide is the ability for network administrators to intervene in the automatic learning and classification process to change the allocation of (p, n)-grams to queues. This would allow administrators to enforce certain policies in cases where the automatic learning and classification process contradicts established service provider's policies.

Despite the number and magnitude of questions that remain unanswered, we are optimistic that diversity-based traffic management holds the promise to address problems that other approaches cannot. As a community, we will have to develop strategies for managing overly aggressive bandwidth consumers within an environment where nobody has complete control over the hardware, network software, or applications; at the same time, though, we cannot make *a priori* decisions as to what traffic patterns and uses are permissible without jeopardizing the very flexibility that has fueled the net's growth. The Internet needs to be able to automatically and autonomously balance the myriad demands placed upon it by both malicious and non-malicious applications. We believe that our (p, n)-gram approach to diversity-based traffic management is a small step towards meeting this need.

Acknowledgments. We thank Evan Hughes and James Kelly, especially for their implementation and analysis work related to (p, n)-grams. We thank Miguel Vargas Martin and Tao Wan for their many contributions to this project, including work on earlier reports. We also thank Jean-Marc Robert and Emanuele Jones, for their guidance, many helpful discussions, and literature references.

We would like to acknowledge Alcatel and Canada's MITACS program for funding the initial stages of this project. We also wish to acknowledge Canada's National Sciences and Engineering Research Council (NSERC) for providing all three authors with NSERC Discovery Grants. The second author also wishes to acknowledge NSERC for funding his Canada Research Chair in Network and Software Security.

References

1. Staniford, S., Moore, D., Paxson, V., Weaver, N.: The Top Speed of Flash Worms. In: Proceedings of ACM Workshop on Rapid Malcode (WORM). (2004)
2. Floyd, S., K.Fall: Promoting the Use of End-to-End Congestion Control in the Internet. IEEE/ACM Transactions on Networking **7** (1999) 458–472
3. Widmer, J., Denda, R., Mauve, M.: A Survey of TCP-Friendly Congestion Control. IEEE Network (2001)
4. Matrawy, A., Lambadaris, I.: A Survey of Congestion Control Schemes for Multicast Video Applications. IEEE Communications Surveys and Tutorials **5** (2003) 22–31
5. Floyd, S.: TCP and Explicit Congestion Notification. ACM Computer Communications Review (1994) 10–23

6. Floyd, S., Jacobson, V.: Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Trans. on Networking (1993) 397–413
7. Estan, C., Varghese, G.: New Directions in Traffic Measurement and Accounting: Focusing on Elephants, Ignoring the Mice. ACM Trans. on Computer Systems **21** (2003) 270–313
8. Estan, C., Savage, S., Varghese, G.: Automatically Inferring Patterns of Resource Consumption in Network Traffic. In: Proceedings of ACM SIGCOMM'03, Germany (2003) 270–313
9. Clark, D., Fangand, W.: Explicit Allocation of Best Effort Packet Delivery Service. IEEE/ACM Trans. on Networking **6** (1988) 362–373
10. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services. RFC 2475 (1988)
11. Georgiadis, L., Guérin, R., Peris, V., Sivarajan, K.N.: Efficient Network QoS Provisioning Based on Per-Node Traffic Shaping. IEEE/ACM Transactions on Networking **4** (1996) 482–501
12. Elwalid, A., Mitra, D.: Traffic Shaping at a Network Node: Theory, Optimum Design, Admission Control. In: Proceedings of IEEE InfoCom'97. (1997)
13. Mahajan, R., Bellovin, S., Floyd, S., Ioannidis, J., Paxson, V., Shenker, S.: Controlling High Bandwidth Aggregates in the Network. Computer Communications Review (2002)
14. Ioannidis, J., Bellovin, S.: Implementing Pushback: Router-based Defense against DDoS Attacks. In: Proceedings of NDSS'02. (2001)
15. Yaar, A., Perrig, A., Song, D.X.: SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In: IEEE Symposium on Security and Privacy. (2004)
16. Heberlein, L., Dias, G., Levitt, K., Mukherjee, B., Wood, J., Wolber, D.: A Network Security Monitor. In: Proceedings of the IEEE Symposium on Security and Privacy. (1990)
17. Hofmeyr, S.: An Immunological Model of Distributed Detection and its Application to Computer Security. PhD thesis, University of New Mexico (1999)
18. Singh, S., Estan, C., Varghese, G., Savage, S.: The EarlyBird System for Real-time Detection of Unknown Worms. Technical report - cs2003-0761, UCSD (2003)
19. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated Worm Fingerprinting. In: Proceedings of OSDI '04, San Francisco CA (2004)
20. Rabin, M.: Fingerprinting by Random Polynomials. Technical report 15-81, Harvard University (1981)
21. Kreibich, C., Crowcroft, J.: Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In: Proceedings of HOTNETS-II. (2003)
22. Kim, H., Karp, B.: Autograph: Toward Automated, Distributed Worm Signature Detection. In: Proceedings of the 13th Usenix Security Symposium. (2004)
23. Wang, K., Stolfo, S.J.: Anomalous Payload-based Network Intrusion Detection. In: Proceedings of RAID'04. (2004)
24. Mahalanobis, P.: On the generalized distance in statistics. Proc. Natl. Institute of Science of India **2** (1936)
25. Staniford, S., Paxson, V., Weaver, N.: How to Own the Internet in Your Spare Time. In: Proceedings of the 11th USENIX Security Symposium. (2002)
26. Matrawy, A., Somayaji, A., van Oorschot, P.: The Threat of Attacker Innovation to Flash Worm Defenses. Manuscript in Preparation (2005)
27. Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., Weaver, N.: The Spread of the Sapphire/Slammer Worm. Technical report, CAIDA et al. (2003) http://www.caida.org/analysis/security/sapphire/.