Preventing Denial-of-Service Attacks with Packet Symmetry

by

Mike Wood

B.Math, The University of Waterloo, 2005

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University of British Columbia

September, 2007

Abstract

Denial-of-service (DoS) attacks are a serious problem affecting the Internet today with security firms estimating over 5000 attacks are launched per day, leading to revenue loss and tarnished reputations for online businesses. These attacks remain prevalent and successful because the Internet has no mechanism to distinguish wanted from unwanted packets. The core of the Internet impartially forwards any packet to its destination without regard as to whether the destination actually desires said packet or not.

This thesis evaluates packet symmetry [47] as a heuristic to distinguish wanted from unwanted traffic at the source network, to enable proactive filtering of DoS attack traffic before it reaches the core. Packet symmetry measures the "goodness" of outgoing traffic using the ratio of transmitted-to-reply packets with a lower ratio implying better traffic. A packet symmetry limiter shapes outgoing traffic to ensure the per-flow ratio of transmission-to-reply packets never exceeds a pre-defined threshold. This empowers DoS victims to throttle any unwanted traffic from symmetry-limited sources simply by not replying to those sources' requests. This power is especially important for end users and small businesses, who make up the majority of DoS attack victims [56, 53], that cannot afford to over-provision network resources as a means to tolerate massive flooding attacks. The net effect is that a network governed by packet symmetry cannot be the source of flood-

ing DoS attacks, as senders are automatically rate-limited proportional to the rate of reply. In this thesis, analysis of network traces helps derive packet symmetry limiting principles and thresholds that effectively discern innocent from malicious DoS traffic with few false-positives. The implementation of a symmetry limiter prototype for the Linux kernel and corresponding deployment on a UBC research lab network evaluate the efficacy of the solution on live traffic with encouraging performance and usability results.

Table of Contents

List of Figures

Acknowledgements

Tremendous thanks to Andrew Warfield and William Aiello for all their dedication, insight and direction.

Many thanks to Christian Kreibich and Vern Paxson for lending network traces, sharing Bro expertise, and assisting with packet dump analysis.

A big thank you to Michael Sanderson for all his help coordinating the deployment on the DSG subnet, and for tolerating my inexperience as a rookie sysadmin.

As well, thanks to Nels Anderson, Aiman Erbad, and Kan Cai for bravely volunteering their machines to participate in our live deployment.

Chapter 1

Introduction

1.1 Motivation

Denial of service (DoS) attacks are a major problem affecting the Internet today with a recent security report estimating that an average of 5,213 attacks are launched per day [28]. These attacks are part of a growing cybercrime industry, in which DoS attacks are used to extort money from legitimate [11] (and illegitimate [14]) businesses. At the extreme, persistent DoS attacks can even put a company right out of business, as was the case for the anti-spam firm Blue Security [13]. Early DoS attacks in February 2000 against Yahoo!, Amazon and eBay [33] received significant media attention, and seemed to trigger the adoption of over-provisioning network resources as a tactic to simply tolerate flooding attacks. As early as June of that year, firm guidelines were set out to protect the critical DNS root name servers from DoS floods, requiring each server have the capacity to sustain three times its normal peak-load [19]. The failed DoS attacks against the DNS root name servers in 2002 [30] and again in 2007 [39] demonstrate how over-provisioning was and has remained a largely successful defense – but only for those that can afford it. Consequently, the vast majority of DoS attack victims are small companies, educational institutions, and government sites [56, 53] that lack the funds to over-provision network resources to resist DoS

floods, making them the most vulnerable to attacks.

It is well understood in the computer security community that a significant amount of computer crime, DoS attacks included, is carried out through the use of botnets – groups of compromised "zombie" computers that are controlled by a remote master machine. Both the academic [10] and industrial [28] communities report on the extremity of botnet infestation in the Internet, with the number of compromised machines in the millions and the number of active malicious hosts per day over fifty thousand. Making matters worse, a botnet does not require a large number of zombies to be capable of launching a powerful attack. A relatively small botnet of only a thousand nodes with an average uplink bandwidth 128 Kbps can offer an attack bandwidth exceeding 100 Mbps, capable of saturating the link to even a modest sized corporation [10]. Notice that no one machine is responsible for generating a large portion of attack traffic – the strength comes from the aggregation of the small amount of attack bandwidth offered by each zombie. DNS amplification attacks are even more potent, as each attack packet from a zombie machine triggers a name server to send multiple flooding packets to the target victim, generating aggregate attack strengths exceeding 7 Gbps [71].

The threat is real and the attacks are damaging, proving costly for both the source and destination networks. The targeted service is degraded or unavailable, leading to revenue loss and a damaged reputation. The source network wastes valuable bandwidth on attack traffic, requiring edge ISPs to purchase larger capacity links from upstream carriers. Both ends may have legal fees surrounding liability issues, as expertise from the legal community [66, 44] suggests that further changes to computer crime laws will hold the owners of zombie machines responsible for the damages from attacks

perpetrated by their machines. Such legislation may deem a corporation or end-user negligent for lacking to implement computer security best-practices – forcing the owner to take responsibility for *all* actions carried out by their machines, not only the actions that were intended.

1.2 Self-critical networks

We introduce the concept of a ***self-critical network*** that filters its own outgoing traffic, taking responsibility for its own actions and preventing its clients from polluting the external network with malicious traffic. Filtering in this manner can be highly effective, as the filtering mechanism is in close proximity to the source of the malicious traffic to be filtered. A number of technical advantages stem from this characteristic. Most importantly, the filtering mechanism is close enough to the traffic source to enforce source address integrity. This ensures further filtering occurs with high fidelity as a source cannot masquerade as another to hide its malicious behaviour. The source network also has complete isolated administrative control over filtering policy and enforcement – no collaboration with other networks is required for a network to police itself. Self-critical filtering is thus immediately deployable, with only the deploying organization having to generate the necessary political momentum and effort from within.

Today, a number of network properties are already enforced in a self-critical manner, though none that directly combat DoS attack traffic. Ingress filtering [31] prevents an end host from forging (spoofing) its source IP address. Contrary to popular belief, ingress filtering is quite widely deployed, as the Spoofer project [15] estimates that less than seventeen percent of net blocks are spoofable, indicating that many edge networks filter packets with

invalid source addresses. Self-critical filtering is also being used to combat email spam. AOL, one of the largest North American service providers, only allows its subscribers to send email traffic (on port 25) that passes through an AOL-managed email server – all other email traffic is dropped [7]. Additionally, network management equipment vendors (ex. [51]) sell monitoring and security products for ISPs that analyze and reduce the unwanted traffic from the ISP's network, which indicates there is an existing market for upstream self-criticizing firewall products. Thus, source-based filtering may serve as a key component in the battle against DoS attacks, if a self-critical DoS mitigation solution can be made practical.

Self-critical filtering of malicious DoS traffic at the source network could ultimately lead to a stronger and safer Internet. However, important questions remain, such as: What exactly should a provider filter at their network? What heuristic can be used to flag malicious traffic? How does the heuristic cope without being able to observe all the aggregate attack traffic? Can such a heuristic reduce the amount of malicious traffic coming out of their networks *and* continue to support network and protocol innovation? What incentives could drive ISPs to invest in a source-based solution, as the filtering does not directly benefit (and may even disturb) its own subscribers? This thesis considers these questions with respect to preventing DoS activity.

1.3 Preventing DoS at the source

DoS attacks remain prevalent and successful simply due to the fact that the Internet lacks a definition or mechanism to discern wanted from unwanted traffic. An end host has no means to specify that it does or does not wish

to receive requests from another host. Network routing is based solely on best-effort packet forwarding – a router looks at the destination address of a packet and forwards the packet along the path towards the destination, without any regard as to whether the destination *actually wants* said packet or not.

This thesis further evaluates packet symmetry [47] as a heuristic to measure the relative *goodness* of network traffic to thwart DoS attacks. The heuristic defines good traffic as symmetric traffic, comparing the number of packets flowing in both directions – source to destination and destination back to source – with better traffic having a more balanced exchange of packets between the two endpoints. Packet symmetry makes the act of communicating a necessarily cooperative endeavour, since both ends must participate equally to maintain the goodness of the packet exchange. In this manner, packets from one end can be thought of as "credits" for the other end to send more packets back. Reply packets from an destination end host are interpreted as signaling to indicate that the destination is willing to receive more traffic from the source. Conversely, a destination can throttle undesired traffic from (potentially) malicious hosts by simply not replying to those hosts' requests.

The net effect is that a network governed by packet symmetry cannot be the source of flooding DoS attacks, as senders are automatically rate-limited proportional to the rate of reply. For a flood victim, no reply traffic gets through their congested link, which in turn causes all symmetry-limited subscribers to be throttled, directly combatting the attack. A symmetry limiter need not observe all the attack traffic to detect attacks. Malice is implicitly inferred with a lack of reply traffic from the destination host. As such, end hosts are automatically protected from bandwidth flooding attacks

perpetrated by symmetry-limited botnets, as each bot can no longer send an asymmetric flow of bandwidth consuming data to the destination without explicit reply from that target.

DoS attacks are not a recent invention – the first large-scale attacks occurred in 2000 [33] and many solutions have been proposed to combat DoS activity since then. However, rate limiting traffic with packet symmetry has a number of advantages over previous work. Firstly, packet symmetry captures implicit signaling already present in the communication patterns of most network applications. Protocols and applications do not need to change to explicitly maintain symmetry, it already occurs naturally, which makes deployment less invasive. Prior work tends to introduce explicit signaling into the network, forcing routers and/or end-hosts to change existing protocols or install new ones, which drastically increases the cost of deployment. Furthermore, source-based packet symmetry filtering is a proactive solution, preventing flooding attacks from happening in the first place, while a large portion of prior work is reactive, detecting and then combatting attacks. Packet symmetry limiters also empower DoS victims to thwart arbitrary resource exhaustion attacks, as the victim can control the attackers' flood rate by manipulating their own reply rate, which can be as simple as inserting a firewall rule to drop packets from the attack sources. Such fine-grained autonomous control over DoS defense is key for smaller institutions, which, as noted, comprise the majority of attack victims. Previous work also tends to focus on high-rate in-core attacks, using probabilistic and statistical analysis on traffic aggregates to detect DoS flooding, which can often be subverted by sophisticated low-rate attacks. Packet symmetry also provides defense against these sophisticated attacks, with the full comparison given in Chapter 5.

As discussed in [32], a DoS defense solution must make both technical and economic sense, otherwise the solution will not be widely adopted. A key element of sensible solutions is minimal invasiveness, requiring no changes to end hosts and little modification to routing infrastructure. Packet symmetry goes one step further, without any need to change end hosts or routers, and without need for new protocols or modifications to existing protocols. Thus, packet symmetry filtering is technically feasible, enabling *practical* incremental deployment with increased benefit in the reduction of DoS traffic for each and every deployment. However, the economic incentive to motivate ISPs to deploy self-critical symmetry limiting is less concrete. Symmetry limiting would reduce the upstream bandwidth consumed on the ISP's peering and upper-tier links, potentially reducing usage costs. However, the economic gain would depend on the proportion of DoS attack traffic contributing to the ISP's peak load under the 95^{th} percentile charging model (a common pricing model for inter-ISP relationships [76]), which may not be very significant, especially for large networks. A stronger potential incentive for symmetry limiting has both the subscribers and ISP alleviated from legal responsibility for DoS activity generated by symmetry limited machines, as the ISP could not be deemed negligent assuming symmetry limiting were to become a network security "best practice" [66]. However, computer crime legislation does not specifically make DoS attacks illegal [27], which prevents prosecutors from securing convictions even when the perpetrators can be identified [41]. Disappointingly, calls to amend legislation to outlaw DoS attacks have been lingering since the major attacks in 2000 [44, 52], which suggests expectations for imminent legislative change may be overly optimistic.

Lacking concrete incentive for deployment, this thesis sets out to bolster

the technical argument in favour of self-criticizing DoS prevention through source-based packet symmetry limiting. The contributions of this thesis include; (i) the derivation of a packet symmetry metric and accompanying limiting thresholds that effectively discern wanted from unwanted flooding traffic with few false positives; (ii) the measurement of the protection provided by symmetry limiting, demonstrating its effectiveness against modern DoS floods; and (iii) the construction of a symmetry limiter prototype and corresponding live deployment, highlighting the feasibility (both for performance and usability) of the approach. In closing, the thesis includes discussion on the potential role of symmetry limiting in the current and future Internet architectures.

Chapter 2

Packet Symmetry Principles and Design

Packet symmetry defines a property of "good" network traffic as traffic that corresponds to a conversation between two (or more) *consenting* hosts. Denial-of-service (DoS) attacks remain prevalent and successful simply because the network lacks such a definition. At present, the network impartially forwards any packet to its specified destination, without regard as to whether the destination actually desires said packet or not.

Specifically, packet symmetry defines the goodness of traffic as the symmetry between packets flowing from source-to-destination and destination-to-source. Reply packets from a destination host to a source host are interpreted as signaling by the destination that it is willing to receive more data from the source. A destination host can throttle any traffic from unwanted or malicious sources simply by not replying to those sources' requests.

The interpretation of reply packets as permission, given by the destination to the source to send more data, captures implicit signaling that already exists in network traffic. Alternate proposals consider making signaling explicit with new cryptographically secure DoS-resistant end-to-end protocols. However, these proposals are not incrementally deployable as both the source and destination ends must implement the new protocol. Us-

ing the implicit signaling already present in network traffic dynamics allows a packet-symmetry DoS solution to be incrementally deployed at various sites - strictly increasing the strength and value of the network with each and every deployment.

A symmetry limiter is most effectively deployed at the source Internet service provider (ISP), as a mechanism to rate-limit the outgoing traffic of its own subscribers. The source ISP is in the best position to filter this traffic, as it is close enough to ensure the source address integrity of its subscribers, leading to high fidelity filtering. Packet symmetry also solves the problem that not all attack traffic can be observed at the source network, since DoS attacks can be widely distributed across networks of zombie computers (botnets). The symmetry limiter infers a destination is under attack when a lack of symmetry, or lack of reply packets from a destination, is observed. This strategy has the added benefit that a successful DoS attack will cut-off the attack traffic, as a lack of replies from the victim will shutdown any symmetry-limited attackers.

2.1 What is Packet Symmetry?

Packet symmetry measures the balance of network packets flowing in both directions of a connection, rather than measuring the raw bytes being exchanged. All packets are considered equal, meaning a large 1500 byte frame, a small 40 byte frame, and anything in between are all counted as a single unit. Transport and application protocols can thus achieve a high degree of packet symmetry, while maintaining a highly asymmetric data transfer. The key insight is that two willing participants can easily balance the flow of packets between them, whereas if the desire to communicate is not mutual,

the flow of packets is inherently imbalanced. A packet symmetry monitor can independently observe this natural feedback loop of packet exchange between two end points, and judge whether each end point desires more traffic from the opposite end.

In practice, TCP – the most widely used transport protocol on the Internet – has packet symmetry built into the definition of the protocol. TCP is designed to guarantee data delivery across an unreliable (or lossy) channel while maintaining fairness with respect to network usage and congestion. As such, the core algorithms of TCP strongly support packet symmetry of TCP flows. Firstly, acknowledgement (ACK) reply packets give feedback to the sender regarding how much data has been successfully received. Although ACK packets are cumulative, the TCP standard [3] specifies that one ACK should correspond to no more than two data packets – essentially mandating TCP have a packet symmetry ratio of 2:1. For higher flow throughput, a TCP sender can send a burst of data packets at one time, and wait for the corresponding ACKs to be returned. The burst size is defined as the congestion window, and TCP's slow-start algorithm ensures this window size is increased symmetrically. The initial window size is a single packet, from where the window size gradually ramps up as long as the ACKs coming back do not indicate a data loss or other congestion event. Thus, with such strong fairness and symmetric principles inherent in the protocol design, it is expected that all services over TCP will be extremely symmetric and will not be exposed to symmetry limiting.

2.1.1 Tracking Packet Symmetry

The packet symmetry for each network conversation, or *flow*, is tracked separately from all other flows in the network. Let a ***flow*** be defined as a five-tuple of fields from both the IP and transport layer protocols,

(sip, dip, proto, dport, sport)

where each field is represented as follows.

sip	the source IP address
dip	the destination IP address
proto	the IP protocol number, specifying the transport layer protocol
dport	the destination port (if proto is TCP or UDP)
sport	the source port (if proto is TCP or UDP)

Using both the source and destination ports allows a pair of end hosts to maintain multiple different flows with separate packet symmetry dynamics. Separating flows to this degree is necessary to prevent an attacker from flooding a service on an end host by maintaining a high rate symmetric flow to another service on that same end host.

However, tracking packets only at the full five-tuple granularity allows the port number and IP protocol number fields to become DoS attack vectors. For both TCP and UDP, a port number is a 16 bit value, and in IPv4, the IP protocol number of an 8 bit value. Thus, an attacker could flood an end host with a large number of low-rate flows by permuting the $16 + 16 + 8 = 40$ bits from these three fields, fabricating an enormous 2^{40} different flows.

To prevent such attacks, flows are tracked at aggregate granularities as well. The aggregate granularities are ordered from finest to coarsest in the

following table.

flow tuple	*finest traffic granularity*	*aggregates traffic for...*
5	(sip, dip, proto, dport, sport)	one flow to one service at one destination
4	(sip, dip, proto, dport, *)	all flows to one service at one destination
3	(sip, dip, proto, *, *)	all flows per transport protocol at one destination
2	(sip, dip, *, *, *)	all flows to one destination
1	(sip, *, *, *, *)	all flows
	coarsest traffic granularity	

This ordering of finest to coarsest granularity gradually groups related flows together to provide the subsequent limiting algorithm with several intermediate stages at which traffic may be rate limited, in an effort to minimize the collateral damage on innocent flows. For instance, a client with several malicious (i.e. asymmetric) flows to a web server will exhibit significant asymmetry at the 5- and 4-tuple flow granularities with *dport* equal to 80, which will allow symmetry limiting to focus on specifically those records and will allow other innocent traffic (perhaps even to the same destination) to avoid collateral damage limiting as a result of the malicious flows. The limiting algorithm and collateral damage is further discussed in Section 2.2.

Though the 1-tuple does not provide flooding defense for any particular destination, asymmetric traffic at the 1-tuple granularity is indicative of network scanning or probing behaviour. Such scanning may or may not be desirable, so the 1-tuple granularity is included for completeness, to help combat scanning activity if needed.

Tracking Window History

Each flow-tuple granularity has a distinct pair of (tx, rx) counters to track transmission and reply packets separately at that granularity. These counters are used to compute packet symmetry at each of the individual granularities.

The (tx, rx) flow counters are maintained across a discrete time window W, purging older data as time moves forward. The window shifts in smaller discrete intervals of length l, purging the oldest interval of length l to make room to include the current l length period. For enhanced security, the window is constantly updated with the arrival of *every* new packet to ensure the tracking data reflects the most up-to-date packet dynamics of the flows being tracked. When a new packet arrives, the notion of a flow's global time t is updated as the arrival time of that packet and the tracking window is extended to include this newly arrived packet, such that the window length never exceeds W. More precisely, for a packet arrival at global time t, the tracking window includes all packets that arrived in the window

$$[t - (t \bmod l) - (W - l), t].$$

Figure 2.1 illustrates how the window shifts as new packets arrive, with old intervals purged in discrete chunks of length l and the new interval extended smoothly as global time moves forward.

Since the window is meant to track per-flow packet symmetry, the length of l should be set long enough to give a reasonable chance that both a request packet and corresponding reply packet can be tracked within the same window interval. This makes the purging of flow history more stable as a relatively balanced number of requests and replies will be discarded as older intervals are shifted out – maintaining a more steady symmetry value

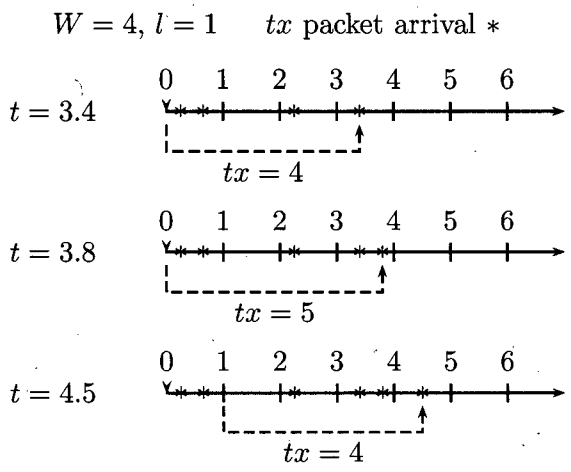


Figure 2.1: Tracking tx packets across a sliding window W.

across the entire window. A typical international round-trip-time (RTT) is on the order of 300-500ms. Thus, setting $l = 2RTT = 1000ms = 1sec$ provides even trans-continental traffic with a very good opportunity to establish symmetry within a single window interval, with a modest cushion to accommodate arbitrary server delay.

2.1.2 Packet Symmetry Metric

Given the (tx, rx) packet counts for a flow, the symmetry s of the flow is computed as the rate of transmission (tx) packets to a single reply (rx) packet. This ratio metric is a natural way to reason about the balance, or imbalance, of packets coming from the source relative to the packets coming from the destination. To accommodate for zero-values, s is computed as:

$$s = \frac{\max(tx,1)}{\max(rx,1)}$$

A symmetry-limited flow is rate limited, either by delaying or dropping transmitted packets, to prevent the symmetry s of the flow from exceeding a maximum threshold X.

The original packet symmetry work [47] proposed a logarithmic metric with $\ln\left(\frac{tx+1}{rx+1}\right)$. However, this metric further exaggerates the symmetry for lower-rate flows by adding the $+1$ to both counters to accommodate for zero values. For example, a flow with $tx = 10$ and $rx = 1$ will have its logarithmic symmetry chopped in half by adding the $+1$ to the rx packet count. While the logarithmic metric succeeds in dampening the effects of larger packet counts, a simple modification to the ratio metric to divide by the smaller of the packet counts avoids any overflow or underflow errors. This modified ratio metric marks the result as positive or negative to indicate which of the transmission or reply directions were larger, thereby communicating the direction of the asymmetric imbalance. This modified ratio metric is computed as:

$$s' = \begin{cases} \frac{\max(tx,1)}{\max(rx,1)} & \text{if } tx \geq rx \\ -\frac{\max(rx,1)}{\max(tx,1)} & \text{if } tx < rx \end{cases}$$

Note that to limit both tx and rx packet symmetry, a limiter using the modified ratio metric must delay or drop packets to ensure that $-X \leq s' \leq X$. However, as the focus of this thesis is the prevention of outgoing DoS flooding traffic, the remainder of the thesis assumes the use of this modified ratio metric to limit outgoing tx traffic, and thus the enforcement of only $s' \leq X$.

2.1.3 Security Guarantee – Bootstrapping Packet Rate

Both the symmetry limit, X, and the history window length, W, have implications for the security and usability of symmetry limiting. A small X will have greater impact on asymmetric DoS floods, but is also less forgiving of innocent asymmetric behaviour, such as buggy applications or user errors. A small W refreshes flow history with faster turnover, aiding recovery for innocent asymmetry but strengthening DoS floods as the asymmetry of flooding bursts are forgotten after shorter intervals. Similarly, a larger X is more forgiving but reduces DoS protection. A larger W retains flow history longer, reducing the effectiveness of DoS attack bursts by lengthening the interval between bursts, but similarly lengthens the recovery time for innocent asymmetry. The larger W also strengthens an attack using covert bursts, where the attacker first establishes a high-rate symmetric flow with the target victim, and subsequently uses the high-rate reply traffic from the victim as the "credit" to blast the destination with heavy attack traffic.

To quantify the security versus usability tradeoff for a symmetry limiter configuration, the symmetry limit X and the history window W combine to form the ***bootstrapping packet rate*** (P_B) – the rate of packets per second at which a source is granted without any reply traffic. This value embodies the guarantee of the symmetry limiter, that all outgoing traffic cannot exceed P_B without corresponding reply traffic. Furthermore, it simplifies the above discussion on security versus usability tradeoffs. Attack strength can be calculated directly from multiplying P_B by the number of attack machines, and there is a clear threshold for the tolerance of innocent but asymmetric behaviour.

The bootstrapping throughput rate is calculated as

$$P_B = X/W \quad \text{packets per unit time.}$$

For instance, a history window of $W = 5$ seconds and a symmetry limit of $X = 10$ packets yields a bootstrapping throughput rate of $P_B = 10/5 = 2$ packets per second.

At minimum, a sender must be able to send at least one packet to a new destination in order to bootstrap the communication, since one side must be allowed to "go first". As such, P_B must be strictly greater than zero. While seemingly trivial, this has important implications for the security provided by a symmetry limiter. As P_B must always be greater than zero, symmetry-limited attackers will always have the ability to send at least one attack packet per window length W. This means that a bandwidth flood to saturate a link of a certain capacity will always be possible with enough machines. However, the end goal is that symmetry limiting reduces attack strength to the point that the cost (to the attacker) for establishing a large enough army of zombies to attack with a certain bandwidth greatly outweighs the benefit of launching an attack for such a bandwidth.

2.2 Limiting Flows with Packet Symmetry

Symmetry limiting enforces that the symmetry value s for each flow falls below a predetermined threshold X. Enforcement involves rate-limiting the transmission of a packet, either by delaying the transmission or dropping the packet, on a flow whose $tx : rx$ ratio would exceed the threshold X if the packet in question were allowed to pass through the limiter.

Packets for each flow are tracked at all five granularities - the 1-tuple through the 5-tuple. Each granularity provides valuable information regarding the traffic dynamics originating from the sender, which leads to an impor-

tant question: At which granularity should symmetry limiting be enforced? Choosing to limit all flows only at the 5-tuple granularity would ignore the traffic dynamics encompassed by the aggregate granularities, potentially allowing heavily asymmetric traffic in aggregate. Conversely, choosing to limit all flows across all granularities can unfairly punish innocent (i.e. symmetric) flows due to misbehaving asymmetric flows from the same sender, leading to collateral damage. As such, the challenge for the limiting algorithm is to simultaneously provide strong protection against packet floods while minimizing collateral damage to innocent traffic.

Meeting this challenge, the proposed limiting algorithm uses the different tuple granularities to ensure that all outgoing traffic from a sender does not exceed the $X : 1$ threshold for *genuine* reply traffic from a destination. Recall from Section 2.1.2 that the symmetry ratio metric accounts for a zero rx packet count by assuming a value of one, preventing the metric from being undefined at flow startup. However, when $rx = 0$ symmetry is precisely that – undefined – since there are no reply packets to compare against the tx traffic. With this in mind, the symmetry limiting algorithm works as follows. When a packet arrives, the limiter looks up the 5-tuple flow record for the packet. If the 5-tuple rx value is zero, then the limiter walks up the *entire* tree of aggregate flow tuples, checking if this packet will exceed the symmetry threshold X at each granularity. Otherwise, if the 5-tuple rx value is non-zero (i.e. $rx > 0$), the the limiter just ensures the packet does not exceed the X threshold at the 5-tuple granularity.

This limiting algorithm both maintains the strong security guarantee of the symmetry limiter (P_B), while simultaneously maintaining innocent flows and limiting malicious ones, even when innocent and malicious flows share the same destination. Firstly, notice how a DoS victim is always empowered

to throttle the traffic of any symmetry-limited sender down to P_B, simply by ceasing to send any reply traffic back to the sender. Regardless of any previous flow history, a destination host can cut off all back traffic to a source, and within one period of length at most W, all flows from that source will have $rx = 0$ and thus will be throttled down to P_B at the 2-tuple *(sip,dip,*,*,*)* granularity. Secondly, this algorithm maintains innocent flows, as flows with at least one reply packet in the last W time period are only limited at the 5-tuple granularity. As such, these flows cannot suffer any collateral damage from limiting at aggregate granularities and need only maintain their own individual symmetry below the X threshold.

Figure 2.2 illustrates this algorithm with one innocent web browsing flow *(192.168.0.1, 10.0.0.1, TCP, 80, 9876)* tracked amongst several malicious flows to the same destination. The malicious flows are marked with source and destination ports containing 666. Notice how the 2-tuple *(sip, dip, *, *, *)* granularity ensures that all traffic, innocent and malicious, is governed by the $X : 1$ ratio keeping the sender's outgoing traffic always in direct proportion to the reply traffic from the target host. Further notice that all the outgoing packets for the web download continue to get through the limiter, avoiding all collateral damage to the innocent flow. Only the malicious traffic, having received no reply traffic, is limited at the aggregate flow granularities.

Lastly, notice that port scanning activity is also throttled by this limiting algorithm. Port scanning is an inherently asymmetric behaviour, sending a large number of single-packet requests to many ports on the target host being scanned. The typical few (if any) replies from the target host will result in significant asymmetry at the 2-tuple *(sip,dip,*,*,*)* granularity, resulting in severe limiting thereby drastically slowing the scanning rate effectively down

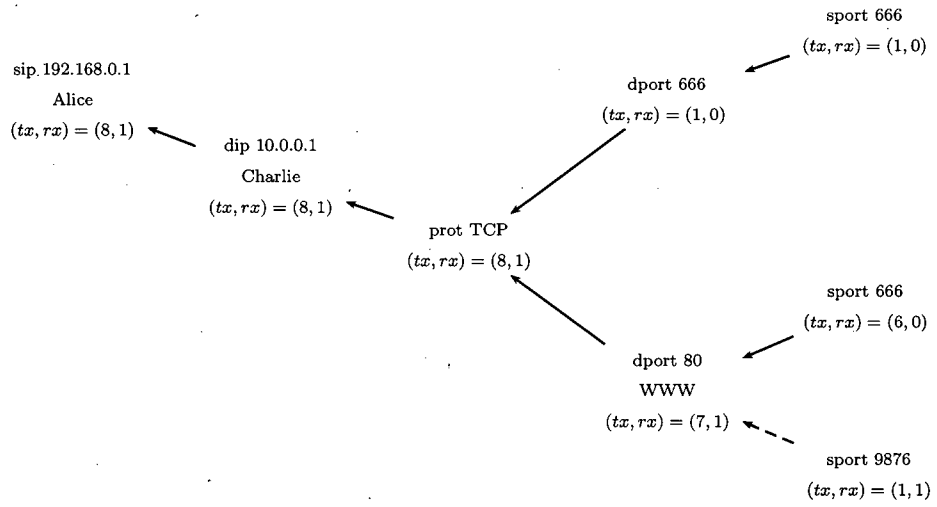


Figure 2.2: Limiting flows with packet symmetry. Assume the symmetry threshold is set at $X = 8$. In this example, packets for the *(192.168.0.1,10.0.0.1,TCP, 80, 9876)* flow are only limited at the 5-tuple granularity, while packets for all other flows are limited at every granularity. For instance, a packet for a new 5-tuple flow for the source-destination pair 192.168.0.1-10.0.0.1 would be rate limited at the 2-tuple granularity, since that granularity has reached the maximum 8:1 threshold for $tx : rx$ packets.

to P_B. Thus, symmetry limiting makes port scanning somewhat redundant, as to achieve a high-rate port scan from behind a symmetry limiter, the scanner must already know the available services of the destination in order to establish symmetric connections to those services to accrue the necessary "credits" to send the scanning packets.

2.3 Where to Place a Symmetry Limiter

Two main factors contribute to the placement of a symmetry limiter in the network; establishing packet provenance and ease of deployment. Packet provenance is important for effective filtering – the more confident it is in the authenticity of packet origin, the more reliable symmetry limiting becomes. The ease of deployment is directly correlated to the technical and economic feasibility of the solution – a solution that is difficult to deploy (technically or financially) is less likely to be accepted.

Considering the points along the path from the subscriber to the ISP, a symmetry limiter could theoretically be deployed at the end-host, at the subscriber's point of connectivity, or at an aggregate point in the ISP infrastructure.

Possibilities for end-host deployment locations include the OS network stack, a virtual machine monitor layer, or the network interface card firmware. While deployment at the end-host guarantees packet provenance, gaining significant deployment penetration can be challenging. Even if widespread deployment could be achieved, persistent and capable computer criminals will likely be able to circumvent the limiting mechanism, even at the virtual machine layer [46]. For example, in Windows XP Service Pack 2 [5] and seemingly in Windows Vista as well, Microsoft introduced a limit on

the number of half-open TCP connections in an effort to combat TCP SYN floods (a common DoS attack [28]). While Microsoft has the market share to achieve widespread deployment for such a mechanism, numerous peer-to-peer sites, such as [57], provide "patches" to subvert these limits. These peer-to-peer applications attempt to open a large number of TCP connections in parallel, which perform poorly if not break under these limits – hence the "patch" to circumvent the protection mechanism at the end host.

The subscriber's point of connectivity, such as the DSL or cable modem, also essentially guarantees packet provenance. As well, the provider has reasonably high assurance that the modem firmware will not be tampered with, meaning the limiting mechanism is unlikely to be circumvented (unlike the end-host deployment discussed in the previous paragraph). Moreover, modern ISPs typically have the infrastructure to automatically upgrade modem firmware and thus could in theory upgrade all their subscribers' modem firmware to perform symmetry limiting. However, having one symmetry limiter per subscriber creates administrative challenges for the ISP. Troubleshooting connectivity with a subscriber may require remote access to the symmetry limiting functionality on the modem, which would then require strong authentication and further complexity built into the modem firmware.

Thus, deployment at a centralized location within the ISP's network seems most pragmatic, making both technical and economic sense. As the limiter moves further from the subscriber, packet provenance is sacrificed but only to the degree of customer aggregation before which address integrity is enforced. Note that address integrity and symmetry limiting enforcement need not occur at the same point along the path. Previously, many ingress filtering deployments prevented spoofing at the network edge, merely ensuring aggregate address integrity for the advertised BGP prefix [16] though

presently the majority of ingress filtering occurs at /24 and /16 network prefix boundaries [15]. To increase the fidelity of packet symmetry filtering, a hybrid architecture might involve upgrading modem firmware to perform the simpler duty of address integrity enforcement at a per-host granularity within the modem itself, with the symmetry limiting mechanism centralized further into the ISP's network. Such a hybrid scheme satisfies both key requirements; (i) enforcing packet provenance with modem firmware guarantees the authenticity of source address information, and (ii) easing deployment with one or a few centralized locations reduces the cost and simplifies the administration of the limiter.

Deploying a symmetry limiter further into the core becomes problematic, as many end-to-end routes on the Internet are asymmetric [58]. A symmetry limiter must be able to see all the traffic to and from the source being limited. If reply traffic can traverse a different route from the transmitted traffic and get missed by the limiter, such flows will be unfairly punished. Indeed, traces of trans-Pacific network traffic from the WIDE project [25] contain many TCP flows which are represented by only a single direction of the communication.

The placement of the limiter does, in part, dictate how traffic can be limited – whether the punishment for asymmetric flows is to delay or drop packets. The original work [47] proposed to buffer and exponentially delay packets for flows exceeding the symmetry threshold. However, this work assumed deployment on the end-host, forcing the burden of buffering packets onto the end-host itself. Unfortunately, for the more administratively attractive network-based deployment, the limiter cannot delay packets since in-network packet buffering is a DoS attack vector itself! Thus, the only practical counter-measure for a network-based symmetry limiter is to drop

packets for flows that exceed the symmetry threshold. As such, the remainder of this thesis assumes symmetry limiting will drop packets for flows that exceed the symmetry threshold.

2.4 Analyzing Symmetry of Internet Protocols

This section analyzes real network traffic traces to determine reasonable values for the symmetry limiting parameters (the asymmetry threshold X, and the window length W), and examines the causes for outlying asymmetric traffic. However, first consider the following taxonomy, represented in Figure 2.3, which outlines how the packet symmetry metric separates all network traffic into classes of "good" traffic and "bad" traffic. Any flow with a symmetry value less than or equal to the symmetry threshold X is hereby blessed as being good, and this traffic remains good as long as sufficient symmetry is maintained, regardless of whether the flow is low or high rate. When a flow's symmetry exceeds the X threshold however, it is then immediately considered a bad or malicious flow and is subject to symmetry limiting. The network trace analysis that quantifies where various types of network traffic fit in this taxonomy is described below.

2.4.1 Data Set

The data set consists of two `tcpdump` network traces. The first is a 24-hour capture from the 100 Mbps link that connects the International Computer Science Institute (ICSI) to the Internet, and contains 21 GB of raw transport and IP layer header data. The second is a 3-day capture from a 100 Mbps link that connects the machines of six Distributed Systems Group (DSG) lab students to the Internet, and contains 248 MB of raw transport and

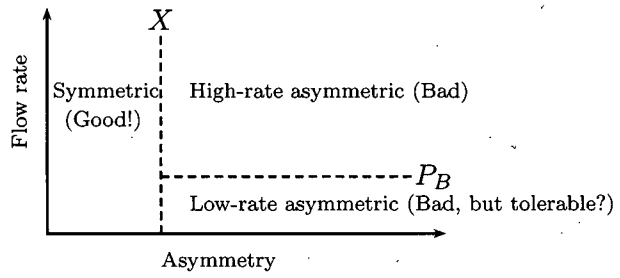


Figure 2.3: Packet symmetry taxonomy for network traffic.

	ICSI		*DSG*	
Total flows	769369	-	27747	-
TCP flows	455352	59.2%	19152	69.0%
UDP flows	265169	34.5%	8430	30.4%
ICMP flows	48848	6.3%	165	0.6%

Figure 2.4: Data set flow distribution by protocol

IP layer header data. Figure 2.4 lists the number of flows in each trace, and further breaks down the total flows by transport layer protocol. The analysis focussed on flows at the 5-tuple granularity, and deemed a flow as finished after two minutes of inactivity.

2.4.2 Traffic Analysis

The analysis was performed with a modified version of Bro [59], using a specialized hook to capture the flow 5-tuple for every packet in a given

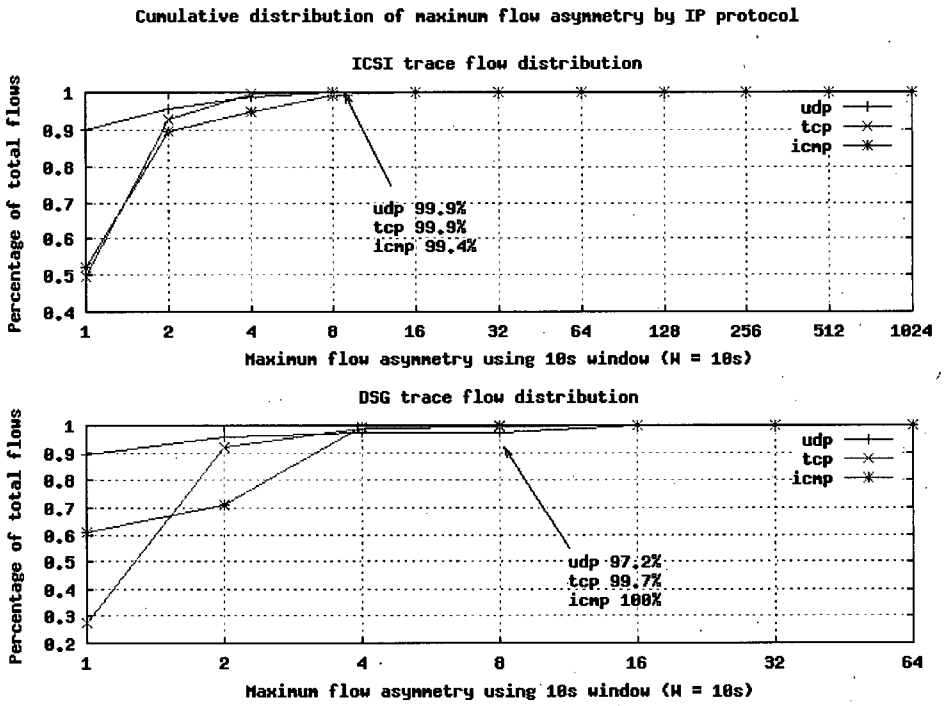


Figure 2.5: Distribution of flows by maximum asymmetry.

`tcpdump` trace. Bro policy scripts were written to track flow symmetry and simulate symmetry limiting against a trace file.

The maximum value of a flow's symmetry is examined first, to determine the proportion of flows that will be exposed to symmetry limiting at any point in their lifetime. Figure 2.5 plots the cumulative fraction of flows with a maximum asymmetry less than or equal to the threshold on the x-axis. This plot indicates that nearly all flows have a maximum asymmetry less than or equal to 8. This suggests choosing $X = 8$ will allow practically all innocent traffic (assuming the traces do not contain a significant quantity of DoS attack traffic).

Choosing a value for the window length W is more complicated than for

the symmetry threshold X, as qualitative factors (ex. the user experience) must be considered alongside quantitative factors (ex. total packet drops). Qualitatively, slow computer response time has been directly correlated to increased end-user frustration [67]. This suggests W should remain small to ensure faster recovery from asymmetric behaviour to prevent frustrated subscribers from switching to a non-symmetry limited ISP. However, both [22] and [42] find that frustration stemming from network usage is largely due to longer download times for web pages with much graphical content. These download times are unlikely to be affected by W, as TCP data transfers are very symmetric. Quantitatively, a larger W increases the protection against DoS floods as increasing W decreases the bootstrapping packet rate P_B, thereby reducing the amount of unacknowledged packets a sender is allowed to produce. These qualitative and quantitative factors illustrate the security versus usability tradeoff of symmetry limiting.

Figure 2.6 simulates symmetry limiting and compares $W = 10$ against $W = 30$, plotting the total number of packets in each flow versus the number of dropped packets for the flow. This plot uses only those flows from each trace whose maximum asymmetry exceeded the $X = 8$ threshold to measure the impact on symmetry-limited flows specifically. As the plot demonstrates, extending the window to $W = 30$ increases the number of dropped packets for most flows. The increase in packet drops for the longer window can be attributed to the increased lag-time for asymmetric behaviour captured at the beginning of the window to be purged. Thus, with the longer window, the chances for end-points to recover from asymmetric communication is hindered, as fewer packets get through to prompt the destination to send replies. Balancing DoS protection and end-user responsiveness, it seems practical to select $W \leq 10$.

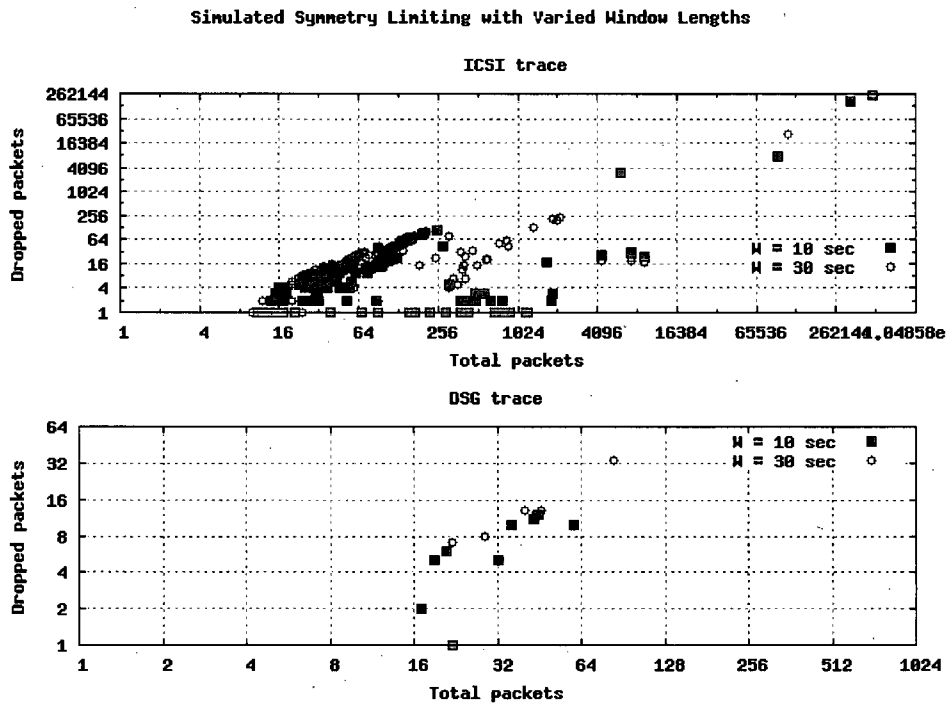


Figure 2.6: Evaluating window history W values.

Recalling that the values for X and W determine the bootstrapping packet rate P_B, it is undesirable to choose a window length W significantly less than the symmetry threshold X, because that will increase P_B and thus increase the attack strength of symmetry-limited DoS floods. Qualitatively, a $P_B = 1$ packet per second seems reasonable for usability, allowing a symmetry-limited host to try to *ping* an unreachable service once per second. Quantitatively, Chapter 3 will evaluate the security properties of symmetry limiting, demonstrating the effectiveness of a $P_B = 1$ packet per second. As such, the results of this analysis suggest the window chosen within the interval $[8, 10]$ seconds.

Having established values for X and W, Figure 2.7 illustrates the dis-

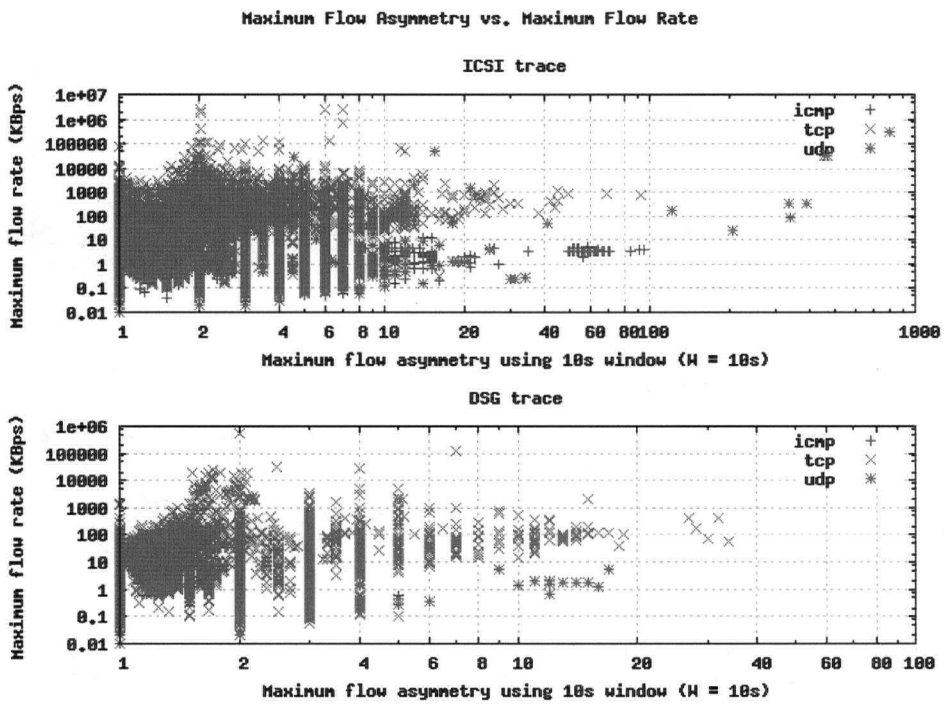


Figure 2.7: Maximum asymmetry vs maximum flow rate by IP protocol

tribution of traffic within the packet symmetry taxonomy discussed above. This plot is encouraging as the majority of network traffic exhibits a high degree of packet symmetry at both low and high throughput rates. Though the plot reveals that a number of flows exceed the $X = 8$ symmetry threshold, recall that from Figure 2.5 these asymmetric flows form a very small portion of the total traffic observed. Nevertheless, the following section analyzes these flows in greater detail to determine the causes for this asymmetric behaviour.

2.4.3 Examining Asymmetric Flows

The variety of network and application protocols is wide and the causes for asymmetric traffic are many, varying with each protocol or application. Though, one general cause of asymmetric behaviour is the metric itself. Packet symmetry is somewhat exaggerated for lower rate flows (i.e. $rx = 1$ or $rx = 2$, a difference of one packet will have a significant effect on the computed symmetry value) which may result in the limiting of relatively low-rate innocent but asymmetric protocols. The following analysis will show that many of the asymmetric flows in the traces reached high asymmetry values due to such low-rate exaggeration. Modifications to the symmetry metric to tolerate such flows range from the naive to the complex. A simple modification uses a fixed constant $c > 1$ rather than 1 when accommodating for zero packet counts, making the new metric $s'' = \frac{\max(tx,c)}{\max(rx,c)}$. A more complex metric might allow short but high rate asymmetric bursts and take packet inter-arrival-times into account. However, any such modification broadens the definition of good (or tolerable) traffic to include greater asymmetry, which directly amplifies the aggregate attack strength of large botnets that simply send the maximum possible number of packets through the limiter. As such metrics strictly reduce the effectiveness of symmetry limiting against DoS attacks, no further exploration of such metrics is undertaken. Lastly, the majority of these low-rate flows are link-local traffic that would not reach an in-network ISP symmetry limiter, which negates the need to modify symmetry limiting to tolerate such traffic.

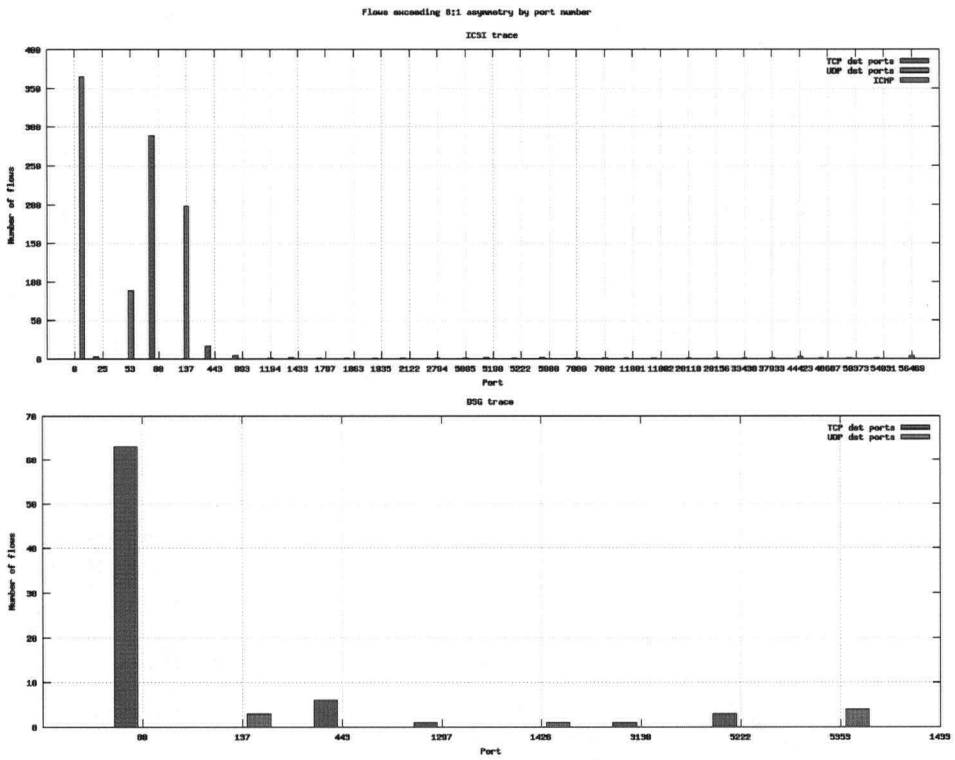


Figure 2.8: Finding ports with the most asymmetric flows.

Transmission Control Protocol (TCP) Outliers

Although TCP is expected to be largely a symmetric protocol, Figure 2.7 does indicate a number of TCP flows whose maximum symmetric metric exceeds the delayed-ACK 2:1 guideline, some of which exceed 10:1 and a very small number that get as high as 30:1. Investigation into these flows yields the following explanations.

Some TCP asymmetry can be anticipated from typical noise associated with communication over a lossy channel. A non-responsive (i.e. failed) endpoint will cause a TCP sender to continuously re-send the most recent unacknowledged packet, doubling the retransmission timeout with each send. A typical RTT would start at 500ms, resulting in unacknowledged packets being sent at 500ms, 1s, 2s, 4s, 8s, 16s and so forth – causing a minor fluctuation in flow asymmetry. ACK loss can also contribute to TCP asymmetry, as earlier ACKs may be lost in transit with later ACKs still carrying the signaling that the earlier data was received. These factors are also exaggerated at lower packet rates, particularly for more recently started flows.

Congestion Window Issues with Idle High-Rate Flows The majority of TCP asymmetry actually stems from a bug in TCP implementations [3]. Reusing a TCP connection after a period of inactivity can cause a TCP sender to inject an unreasonably large congestion-window-sized burst of packets into the network. The fundamental problem is that a previously established congestion window indicates the network congestion at the time the data transfer took place. After a period of inactivity, neither the sender nor receiver have any basis on which to judge the new state of congestion in the network. The TCP standard [3] accounts for this potential issue, stating that after a period of inactivity larger that the retransmission timeout, a

TCP sender should redo slow-start and ramp up its congestion window just as if starting a new connection. However, the bug in many TCP stacks is described well in [3]:

> Using the last time a segment was received to determine whether or not to decrease cwnd fails to deflate cwnd in the common case of persistent HTTP connections [HTH98]. In this case, a WWW server receives a request before transmitting data to the WWW browser. The reception of the request makes the test for an idle connection fail, and allows the TCP to begin transmission with a possibly inappropriately large cwnd.

The vast majority of TCP asymmetry exhibited this traffic pattern, where after a period of inactivity a single request triggered a large burst of data from the opposite direction. Figure 2.8 suggests that most TCP asymmetry is due to this bug, with the most asymmetric flows on port 80 (HTTP) and port 443 (HTTPS). The third most asymmetric TCP port is 993, used for secure email via imap, a similar application in which a pause followed by a request to trigger a large data download would occur, for instance when retrieving new mail.

Drastically increasing W to retain longer flow history would prevent limiting TCP flows in this case. However, a longer window increases the strength of covert burst flooding attacks that establish symmetry and subsequently flood the victim. Computing the symmetry value as a weighted average across different portions of the history window, with the more recent portions more heavily weighted than the older portions, combats such an attack. However, these attacks remain strictly more powerful with the longer window and a weighted average metric than if the window were simply

the most heavily weighted portion of the window with the metric computed across the whole window. All things considered, as the standard indicates, this is fundamentally a problem with TCP implementations, and should be fixed there. In the meantime, a symmetry limiter would serve as a patch for TCP stacks with this bug, as the limiter will simulate network congestion (through limiting the oversized congestion window burst) and thus enforce good behaviour on the flow's behalf. TCP will tolerate and recover from these packet losses, just as it recovers from any other inferred network loss, and will redo slow-start to gradually ramp up its congestion window fairly and representative of the current network state.

Splitting Packets Across Discrete Window Intervals An inherent characteristic of tracking symmetry across a discrete window was found to be another cause for moments of highly asymmetric TCP traffic. The asymmetry stems from the separation of high-rate symmetric request and reply traffic across window interval boundaries, when a significant portion of the reply traffic ends up being tracked in a later interval than the request traffic. When the window interval for the request traffic expires and is purged, the reply traffic remains tracked in the later window interval. At this time, a new low-rate symmetric packet exchange will compute a high asymmetry value, due to the imbalance of ACKs remaining at the tail of the window.

The nature of this problem warrants a shorter window. However, decreasing W increases P_B which means an increase in flooding attack strength. To solve both problems, symmetry can be computed in two stages; the first stage uses a shorter window and is subject to a lower symmetry limit, while the second stage has a longer window with a larger symmetry limit. Figure 2.9 illustrates the process, as symmetry is first computed across W' and is subject to X'. If the flow exceeds X' across W', only then must symmetry

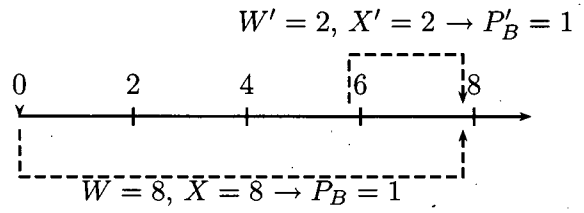


Figure 2.9: Two-stage symmetry computation.

be computed across the full W and compared against X. Notice that P_B remains constant for both stages, maintaining the same security guarantee for the symmetry limiter. Using this two-stage symmetry computation, TCP flows that formerly reached asymmetry values over 15:1 and some as high as 30:1, all returned to the expected 2:1 packet ratio as defined by the TCP standard.

ICMP Outliers

Figure 2.8 indicates a significant portion of asymmetry is due to ICMP flows. Of the 210 total ICMP flows that exceed 8:1 asymmetry, 129 flows are simply due to non-responsive destinations. Of these 129 flows, 118 flows are the result of unidirectional streams of 'destination unreachable' messages, 59 of which come from a single source. The remaining 11 flows consist of unidirectional streams of 'time exceeded' (exclusive) or 'echo-request' messages.

The remaining 81 flows are exchanges between only three external hosts (responsible for 68, 10 and 3 flows each) and various internal ICSI network addresses. The traffic largely consists of 'admin prohibited' messages from ICSI hosts that indicate a particular TCP/UDP host or port is unreachable. This suggests the external hosts are scanning the ICSI network for various

services, perhaps in the hopes of breaking into the network. As such, the ICMP traffic is heavily asymmetric in the direction going from ICSI to the external network, as the single-packet UDP scanning flows that trigger the prohibited messages are tracked separately as UDP flows. The occasional ping request-reply exchange between the external host and ICSI hosts is the reason the flows appeared to have disproportionate but non-zero reply traffic. As such, several of these ICMP flows exceed symmetry of 50:1, with one flow reaching 85:1, and would consequently be subject to severe symmetry limiting.

However, in this case of external hosts port scanning the internal network, severely limiting the host-port unreachable ICMP traffic is in fact quite beneficial. Fewer ICMP packets reduces the effectiveness of the scanner's reconnaissance, as less information propagates back to the attacker. Note that without any limiting, the ICMP return traffic maintains the scanner's symmetry at the 2-tuple granularity, as one UDP scanning packet generates a corresponding ICMP unreachable packet. Thus, consider this scenario with both the scanner and scan-ee behind respective symmetry limiters. As before, the scan-ee's ICMP traffic will be subject to heavy symmetry limiting. This will effectively cut the scanner's scanning rate, as the scanner's outgoing UDP scan will be largely asymmetric since very few UDP replies are returned, and hardly any ICMP unreachable messages are getting through the scan-ee's symmetry limiter. This scenario highlights a cyclic benefit to symmetry limiting, that scanning generates asymmetric return traffic, which will be throttled, causing scanning traffic to become asymmetric, which will then be throttled, reducing scanning rate, and so on.

User Datagram Protocol (UDP) Outliers

NetBIOS asymmetry Figure 2.8 shows port 137 (NetBIOS) is the most asymmetric UDP port. However, a single machine is responsible for 198 of the 199 flows that exceed the 8:1 threshold. These 198 flows consisted of a repeated query to which no response is generated. The one other asymmetric NetBIOS flow seems to represent typical NetBIOS name registration. The flow contained relatively low-rate unicast name registration reports and refresh notices, which largely remained under a 7:1 threshold. Momentary bursts of additional NetBIOS activity resulted in the asymmetry jumping over the 8:1 threshold, with only 2 peaks reaching 10:1, and 9:1, respectively. Figure 2.10 illustrates the evolution of the symmetry value of this flow. Simulating symmetry limiting on this flow with a threshold of 8:1 resulted in a loss of merely 3 packets – all of which were redundant copies of previous requests.

NetBIOS is interesting, as port 137 is commonly used for virus and worm propagation [10]. The ICSI trace reflects this observation, as a single external host was responsible for 514 single-packet flows to various internal ICSI machines. A symmetry limiter would likely reduce worm propagation by drastically slowing the scanning rate to P_B, as the scanning probes tend to elicit few (if any) responses.

DNS asymmetry Only three machines are responsible for producing the majority of the 87 DNS flows exceeding 8:1 asymmetry from the ICSI trace as shown in Figure 2.8, each generating 47, 15, and 14 flows respectively.

A complete lack of reply traffic was the cause of asymmetry for a mere 15 DNS flows. These flows typically involved repeated queries for names

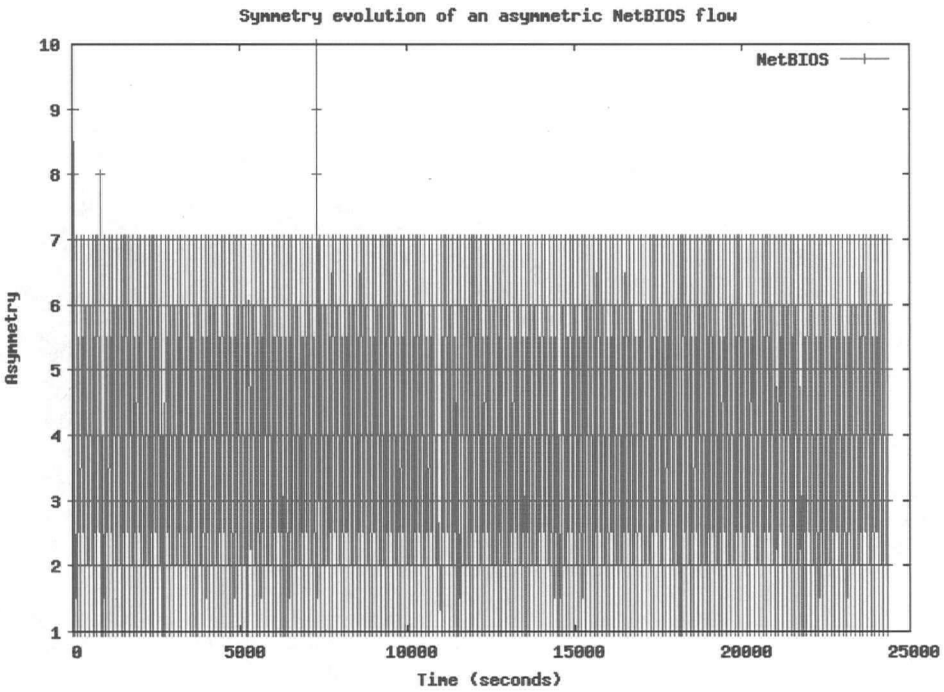


Figure 2.10: A NetBIOS name registration flow.

for which there was no response. Several such queries were lookups for bogon (i.e. unallocated) or blacklisted addresses [26], which is indicative of firewall activity. Note that none of these queries had the recursion desired bit set, which means any reply should have come from the queried destination. This means the asymmetry was simply a result of no response coming back from the queried DNS server, which symmetry limiting would interpret as a success in protecting a DNS server that did not wish to be contacted by the requesting client.

Bursty flows were another cause of DNS asymmetry. A total of 50 asymmetric DNS flows had relatively balanced (tx, rx) total packet counts, however, each flow had a high-rate burst of mixed query and response packets for multiple different DNS names. Such traffic is indicative of exchange between internal ICSI and external DNS servers. These flows suggest that DNS servers should perhaps be exempt from symmetry limiting, due to the lack of flow control to smooth out the bursts. Note that all client end-to-end DNS traffic exhibits symmetry to a high degree, and none of the asymmetric DNS flows are queries from end-hosts. Even more interesting is that not a single one of the machines that produced these asymmetric DNS flows are actually DNS servers. The three machines that produced the most asymmetric DNS flows are file servers in the ICSI infrastructure, and the cause for the abundant DNS traffic is unknown.

The remaining 22 asymmetric DNS flows mainly followed one of two patterns. The first pattern has a single packet query prompt a single packet response from the destination, after which point the source host repeats the query multiple times afterwards, receiving no further response. Symmetry limiting will have little effect on this traffic, since the initial symmetry query-response exchange will succeed, with only later redundant query pack-

ets subject to minor limiting. The second pattern has a single packet query prompt the destination to send a several packet response, causing an imbalance of rx packets over tx packets. Recall from Section 2.1.2 that symmetry limiting does not need to affect this traffic, since the goal is to prevent outgoing (tx) asymmetric behaviour – incoming asymmetric traffic need not be limited.

Multicast DNS asymmetry Multicast DNS is part of the Zeroconf networking initiative [24], providing a link-local namespace in the absence of another naming (i.e. DNS) infrastructure. Applications that share and look for shared content, such as iTunes, make use of multicast DNS to discover the services available on the local link.

Figure 2.8 also shows port 5353 (multicast DNS) had only 5 flows in the DSG trace that exceed the asymmetry threshold, of a total 271 multicast DNS flows. The asymmetry occurs during the startup probing and announcing phase of the multicast DNS protocol, where a host probes for names it would like to register, followed by announcements indicating the host has taken ownership of the names. This startup phase occurs anytime the network configuration or settings may have changed for a host (i.e. ethernet cable plugged in, IP address change, wake up from sleep, etc.). First, note that such local discovery protocols repeat probes/announcements for robustness, as communication via the multicast address (224.0.0.251) is not connection-oriented. A simulation of symmetry limiting on these flows shows that the packets subject to symmetry limiting are simply repeats of previous probes/announcements, meaning that the dropped packets are unlikely to affect the application layer behaviour. Secondly, considering an ISP-managed deployment of a symmetry limiter, it is unlikely such multicast DNS or any other link-local traffic will be subjected to symmetry limiting, as this traffic

should not (by definition) travel further than the local link, which will be contained entirely behind the limiter.

Asymmetry One-offs Figure 2.5 admits that a small number of UDP flows exhibited wildly asymmetric traffic, with asymmetry exceeding 100:1, with one reaching 799:1. This one extremely asymmetric flow is a one-way media stream over RTP, which is a data transfer protocol for real-time applications. RTP separates the control and data channels, using a separate RTSP control protocol that runs over TCP to govern the data transfer that runs over UDP. Indeed, the 799:1 asymmetric UDP flow is a video playback using Windows Media Player, which has a corresponding RTSP flow active throughout the lifetime of the UDP flow.

The separate control and data channels of RTSP and RTP indeed present a genuine case against the deployment of symmetry limiting on every uplink to the internet. However, the threat model targeted by symmetry limiting more seeks to protect such media streaming servers from attacks by malicious clients – not to prevent media streaming servers from launching attacks themselves. Again, the deployment model focusses on limiting outgoing traffic, not incoming. Thus, such high-throughput streaming servers would purchase special non-symmetry-limited connections from their service provider, naturally with additional bookkeeping to prove the legitimacy of the service to prevent attackers from obtaining this powerful non-limited connectivity. Under these assumptions, end hosts will be able to receive asymmetric streams from the media server, but the server will remain protected from malicious clients.

Triangular routing appears to be the cause of another four asymmetric flows, two of which exceed a symmetry threshold of 330:1 and the other two exceed 450:1. However, the four flows seem to be two pairs of related

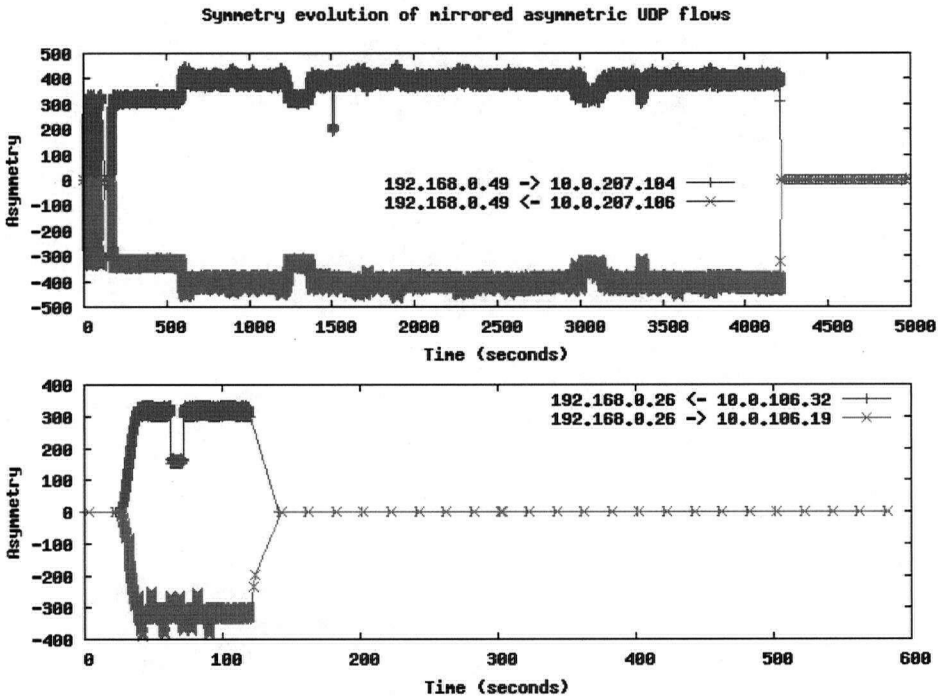


Figure 2.11: Triangular routing of UDP traffic.

flows, forming a triangular routing path. Figure 2.11 plots symmetry over the lifetime of the flows against the actual packet arrival times, showing the number of packets arriving at similar times. Each pair of flows has one highly asymmetric outgoing flow from an ICSI address (198.162.0.0/8) and one highly asymmetric incoming flow from a different external address but coming back to the same internal ICSI address. The similarity in flow symmetry clearly suggests the two flows in each pair are related. Unfortunately, the anonymization of the ICSI trace prevents further investigation into the application(s) involved.

As it stands, a symmetry limiter would severely punish these asymmetric flows that form symmetric triangular routing paths. Assuming a symmetry limiter is only present at one end, the replying host could spoof the source address of the middle-man on the path to make the separate flows appear as a single flow. When all hosts are symmetry limited, all of the high-rate asymmetric traffic will be rate limited, likely preventing these applications from functioning correctly. However, these flows are not representative of the majority of network applications, as these are 4 of a total 769,369 flows observed. But more importantly, such triangularly-routed protocols are fundamentally unfair as without feedback (in the form of acknowledgements) the senders are unresponsive to packet drops due to network congestion.

2.4.4 Considerations for Multicast Protocols

Multicast protocols require special consideration when it comes to symmetry tracking. The reason for this consideration is that a multicast address corresponds to *every host* subscribed to the multicast group associated with that address. Thus, for multicast traffic, there is no longer a one-to-one

mapping between the destination address and recipients of the packet.

A naive but intuitive solution tracks any packet to a multicast address as being received (i.e. an rx packet) by every host on the network, minus the host that sent the packet. While simple, this approach would be rather effective in a symmetry limiting context. Hosts in the multicast group will benefit from the packet being an rx packet on their end, as that will enable such hosts to participate (i.e. reply) to said multicast packet for that specific multicast group. As well, hosts not in the multicast group can simply ignore said packets, with no effect on symmetry limiting since limiting focusses on outgoing tx traffic. Note this solution scales to per-flow tracking, as only one additional symmetry-tracking record per multicast address is needed. In this scheme, each host maintains a flow tuple tracking its own packets sent to the multicast address (just as with any other non-multicast flow). The difference is another global flow-tuple tracks all packets sent to the multicast address. Then, the symmetry for the multicast flow between any one host and the multicast address can be calculated with tx from the host's flow-tuple, and with the rx as the difference of the global multicast address' flow-tuple's rx value and the host's own tx value.

2.5 Summary

This chapter described the principles of packet symmetry and the algorithms used to limit traffic using the packet symmetry metric. The two key parameters for symmetry limiting are the symmetry threshold X, and the tracking window length W. Together, these parameters also form the bootstrapping packet rate P_B, which is the rate at which a sender can transmit packets without receiving any reply. Analysis on real network traffic traces estab-

lished values for these parameters, with $X = 8$ packets and $W \in [8, 10]$ seconds, resulting in a $P_B \leq 1$ packet per second. These values resulted in very few (less than 1%) false positives in the analyzed traffic, the limiting of which would not likely have negatively affected application level behaviour. Deployment targets limiting outgoing tx traffic, which alleviates concerns with limiting false positives due to asymmetric incoming rx traffic as well. Though deploying a limiter at the end host could solve this problem by delaying (and buffering) packets of asymmetric flows, in-network deployment is more manageable, cost-effective and the most unlikely to be circumvented. As such, the limiter must drop (rather than delay) packets to prevent the limiter mechanism from being attacked itself. The limiting algorithm (in Section 2.2) provides the strong guarantee that all traffic from a symmetry-limited sender will always be less than or equal to $X : 1$ in direct proportion to the receiver's reply traffic, and in the case the receiver sends no reply traffic, the sender will not be able to exceed the bootstrapping packet rate P_B. These guarantees afford the opportunity to make strong claims regarding the DoS attack strength capable in a symmetry-limited network.

Chapter 3

Security Evaluation

Symmetry limiting allows an ISP to provide the strong guarantee to the rest of the Internet that no symmetry-limited subscribers can be the source of a DoS flooding attack. Furthermore, with symmetry limiters deployed at the source network governing the network's outgoing traffic, a DoS victim now has the power to control the rate of traffic it receives, simply by manipulating its own reply rate. These are two significant wins for all DoS victims alike, as (i) the overall volume of DoS traffic is drastically reduced and (ii) victims of resource exhaustion DoS attacks no longer have to contact the ISP from which the attack originates and wait for that organization to take action on the victim's behalf – a victim need only insert a firewall rule to drop traffic from the malicious hosts, thereby cutting of all reply traffic to the victim simply by ignoring their incoming requests. This simplicity and localized control over DoS defense is especially important for small businesses and home users, who make up the majority of targeted DoS attack victims [53, 28] and do not typically have the network (or financial) provisioning to tolerate DoS attacks.

Notice how the symmetry-limiting mechanism is designed such that the increasing effectiveness of a flood increases the throttling by the limiter – as fewer replies from the victim can escape the flooded link(s), fewer transmissions are allowed to pass through the limiter due to the lack of

reply traffic. The following sections will establish the threat model and further evaluate the DoS protection provided by symmetry limiting.

3.1 Threat Model

Source-based symmetry limiting is a targeted defense against botnet originated DoS attack activity. It is well understood in the computer security community that a large amount of computer crime is carried out through the use of botnets [10]. A botnet is a group of compromised (exploited) "zombie" computers that can be controlled by a remote master machine. In the last quarter of 2006, Symantec [28] reports observing more than 6,000,000 distinct bots, with an average of just over 60,000 active bots per day. The HoneyNet Project in 2005 [10] reported tracking botnets of varying sizes, from hundreds to hundreds of thousands of nodes.

Defining the botnet as the DoS attack vector allows the threat model to make certain key assumptions about the parameters that determine the strength of attack. Particularly, the botnet master can only enlist zombie machines he or she is able to exploit. Assuming all machines with any hardware or software configuration are equally likely to be exploited, the properties of botnet zombies and their network connectivity can be generalized over global statistics on network connectivity. Note this assumption will likely result in an over-estimate of the attack strength for a symmetry-limited botnet, as it is more likely older and unpatched technologies are more vulnerable to exploitation, the users of which are equally more likely to have lower upload bandwidth. Conversely, corporate users with more powerful uplink speeds tend to fall under the wing of a regulated IT management infrastructure, which will likely have tighter security and thus be

less vulnerable to compromise.

Users of zombie machines are typically oblivious to (ignorant of) the fact the computer is enlisted in a botnet. Moreover, due to the variety of malicious activities an infected machine may inflict on its owner (ex. phishing) [10], it seems extremely unlikely that the machine owner will knowingly take action to contribute to the success of botnet related activities. This leads to the assumption that the zombie machine owner will not attempt to circumvent a symmetry limiting mechanism deployed by the service provider (ex. by tampering with the ISP cable or DSL modem), meaning that all traffic from a symmetry-limited zombie will pass through the symmetry limiter.

Internet connections are typically asymmetric with respect to download and upload speeds, having more bandwidth allocated for download. Large broadband service providers offer connectivity with upload speeds ranging from 64 Kbps to 800 Kbps, and some over 1 Mbps [62, 73]. Maintaining the assumption that all machines are equally likely to be compromised by a botnet master, it is reasonable to assume the average zombie provides the median of the upload speeds, roughly approximated as 384 Kbps.

TCP Reset and TCP FIN floods with packet sizes less than 100 bytes are the most common DoS attacks [53]. DNS amplification attacks [71], like the one used to put the anti-spam company Blue Security out of business [13], similarly use small attack packets, which contain a single DNS query for a large resource record, resulting in an amplified response of several large packets sent to the victim. However, pure bandwidth flood attacks from a symmetry-limited botnet will result in greater damage to a victim than these small packet attacks, as the symmetry limiter is agnostic to the data carried by each packet. Furthermore, the DNS amplification attacks are an example of reflector attacks that employ source IP address spoofing to redirect the

amplified reply to the targeted victim. Such attacks cannot be launched from within a symmetry limited network, recalling the original requirement for a symmetry limiter to have source address integrity. Thus, assuming the attacker will maximize attack strength, the subsequent analysis focusses on attacks using large packets. The ubiquity of Ethernet deployment makes it reasonable to assume the path maximum transfer unit (MTU), and hence the maximum packet size, for zombie machines as 1500 bytes [38].

Many sensible service providers currently deploy ingress filtering [31] for their subscribers. Recent analysis from [15] estimates that, as of June 2007, less than 19% of IP addresses are spoofable and less than 17% of netblock addresses allow spoofing. It seems reasonable to assume that a similar proportion of sensible service providers would be willing to deploy a symmetry limiter to further ensure the integrity of their subscribers' traffic.

3.2 Effectiveness Against DoS attacks

An army of symmetry-limited bots evokes the greatest DoS attack power with a bandwidth flooding attack, as the symmetry limiter simply counts the number of packets and is agnostic to packet data length. The following list formalizes the parameters to compute the DoS flooding attack strength of a given symmetry-limited botnet.

N	100 - 100000	size of botnet (number of controlled machines)
B_{zombie}	384 Kbps	upload bandwidth of controlled zombie machine
P_{attack}	1500 B (11.71 Kb)	length of a bandwidth flood attack packet
L	$> 80\%$	percentage of symmetry-limited zombies
X	8	the symmetry x:1 ratio threshold
W	10 sec	the window history length

3.2.1 Traditional DoS attacks

A naive DoS attacker will simply blast as many large packets as possible at the victim. Considering each machine independently, the number of attack packets allowed through the limiter will be equal to the bootstrapping packet rate (P_B) since the victim is assumed not to reply to the flooding traffic. Thus, the attack strength, A_{flood}, is computed as

$$A_{flood} = P_{attack}P_B(NL) + B_{zombie}(N(1-L))$$

The attack strength equation clearly indicates attack strength is linearly proportional to the botnet size, the proportion of limited vs. non-limited zombies and the symmetry threshold itself. While an exponential decrease in attack strength is desirable, this cannot be achieved when each source host is considered independent of all other sources. Although linear, Figure 3.1 plots the attack strength in Mbps, which highlights the effectiveness of symmetry limiting against naive attackers, showing a significant decrease in aggregate attack bandwidth.

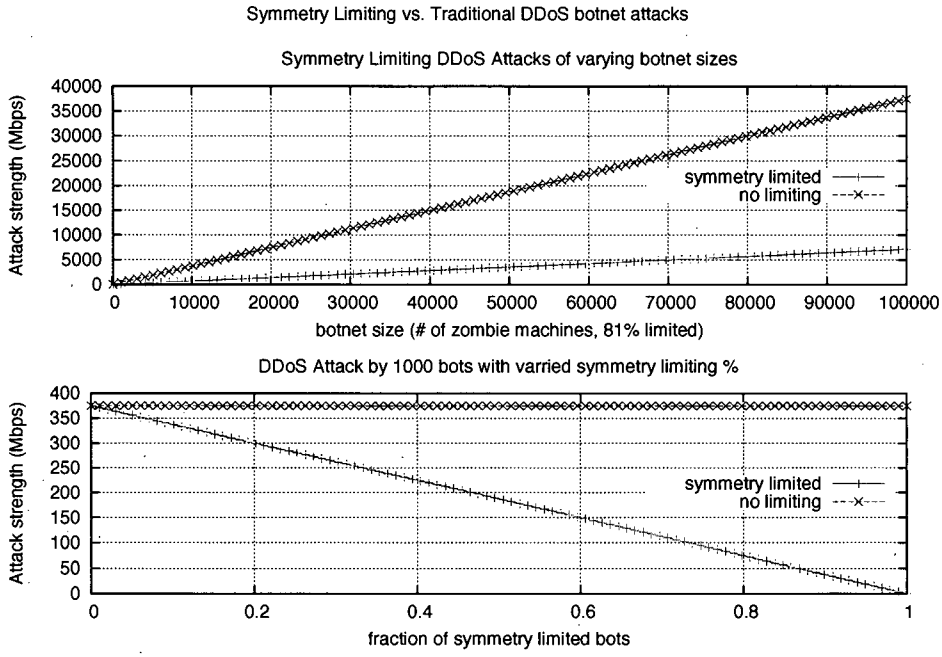


Figure 3.1: Symmetry limiting vs. simple DoS flood

An additional deterrent for botnet masters running DoS attacks with symmetry-limited zombies is that heavily limited machines will reveal the identities of compromised machines. A key advantage for the botnet owner is the stealthly nature of bot activity – the zombie owner is largely unaware of the malicious activity perpetrated by their machine. A symmetry-limiter exposes the identity of bot-infected machines, as periodic blasting of wildly asymmetric traffic is a clear signature of DoS attacks. A symmetry-limiting ISP gains the opportunity to alert the subscriber of infections or to take more drastic action, such as blocking all traffic from the subscriber until he or she cleans up their machine.

3.2.2 Symmetry-Aware DoS Attacks

Covert Bursts A moderately intelligent attacker may attempt a bursty attack - establishing high rate symmetric flow(s) with the victim to build up symmetry "credits" with which a large amount of attack traffic may subsequently be sent. For instance, the attacker might request to download a very large file from the victim to establish a high-rate symmetric data transfer over TCP, after which the malicious sender floods the victim with TCP SYN packets for the same flow 5-tuple(s). In this manner, an attacker can even momentarily exceed the symmetry threshold to at most $2X$, by abusing the limiting algorithm – first using the reply traffic to send flooding packets at aggregate granularities, and then using the same reply traffic to send further flooding packets at the same 5-tuple granularity containing the genuine reply traffic. However, as time moves forward, the previously established symmetry of the flow is lost as the high reply rate is diminished with the success of the flood. Moreover, the attack success prevents symmetry

from being re-established as the victim is unable to send as many replies back to the malicious sender. For every victim, there exists a steady state which maximizes the attack bandwidth while providing the victim with just enough outgoing bandwidth to maintain the symmetry of the attack flow. However, with a symmetry-limiter deployed at the source, the victim is in a strong position to employ counter-measures against such an attack by any customized policies or heuristics with which to control their own reply rate. Thus, for such an attack, the victim has the simple task to detect and combat such attacks by not replying to such malicious senders. Thereby downgrading the strength of the bursty attack to the strength of the pure flooding attack discussed in the previous section.

Collusion A more sophisticated symmetry-aware DoS attacker will attempt to forge reply traffic from the victim to the zombie machines to get more attack packets through the symmetry limiter. An additional parameter for the acknowledgement packet length is required to formalize this type of attack.

P_{ack}	40 B (.31 Kb)	byte length of an acknowledgement (reply) packet

The sophisticated attacker needs to co-ordinate collusion between his or her bots to make sure forged back-traffic is generated from a machine that will be capable of sending such forged packets. Although the attacker has ultimate control over the connectivity of their own machines, directly involving their own machines in the attack traffic stream increases the risk of exposing their true identity on the network, making this option suboptimal for the attacker. Symmetry-limited bots are also presumed to be ingress filtered and consequently cannot send packets with spoofed source address information. This leaves the attacker only the portion of their botnet that

is not symmetry-limited with which to forge victim reply traffic.

A single forged acknowledgement may only contribute to establishing the symmetry of at most one symmetry-limited zombie, as the source IP address field is present in all flow tracking granularities. Thus, the ideal collusion attack employs a greedy algorithm to assign the minimum amount of non-limited bandwidth to generate forged acknowledgements to produce the maximum amount of attack bandwidth from symmetry-limited zombies, and uses any remaining non-limited bandwidth as further attack bandwidth. For the moment, assume the attacker develops a clever distributed algorithm to (i) determine which zombies are and are not symmetry limited, and to (ii) co-ordinate the collusion of non-limited zombies to send forged acknowledgements to disjoint sets of symmetry limited zombies. Further assume that no zombie receives duplicate forged acknowledgements and no forged acknowledgements are wasted by being sent without the possibility for the symmetry-limited zombie to send a greater amount of attack traffic. With these assumptions heavily weighted in the attacker's favour, the attack strength, $A_{collude}$, is computed as

$$\begin{aligned} F_{pot} &= N(1-L) * B_{zombie}/P_{ack} \\ F_{req} &= NL * B_{zombie}/(P_{attack}X) \\ A_{sym} &= P_{attack}X * \min(F_{req}, F_{pot}) \\ A_{free} &= N(1-L) * B_{zombie} - P_{ack} * \min(F_{req}, F_{pot}) \\ A_{collude} &= A_{sym} + A_{free} \end{aligned}$$

where the individual computations represent

F_{pot}	the total number of potential forged acknowledgements, available to be sent by non-symmetry-limited bots
F_{req}	the total number of required forged acknowledgements; needed to saturate the sending links of the symmetry-limited bots with attack traffic
A_{sym}	the aggregate attack strength of the symmetry-limited bots
A_{free}	the aggregate attack strength of the non-symmetry-limited bots
$A_{collude}$	the total aggregate attack strength of the bot net

Figure 3.2 plots the attack strength for this symmetry-aware collusion attack versus an attack of equal size without symmetry limiting. Both the non-limited attack and collusion attack have the same attack strength, with symmetry limiting only becoming effective when the number of potential forged acknowledgements becomes smaller than the number of required forged acknowledgements to saturate the symmetry-limited bots' attack bandwidth. Notice this happens when

$$\begin{aligned} F_{req} &> F_{pot} \\ NL * B_{zombie}/(P_{attack}X) &> N(1-L) * B_{zombie}/P_{ack} \\ \frac{L}{1-L} &> \frac{P_{attack}X}{P_{ack}} \end{aligned}$$

Let $c = \frac{P_{attack}X}{P_{ack}}$, then solving for L ...

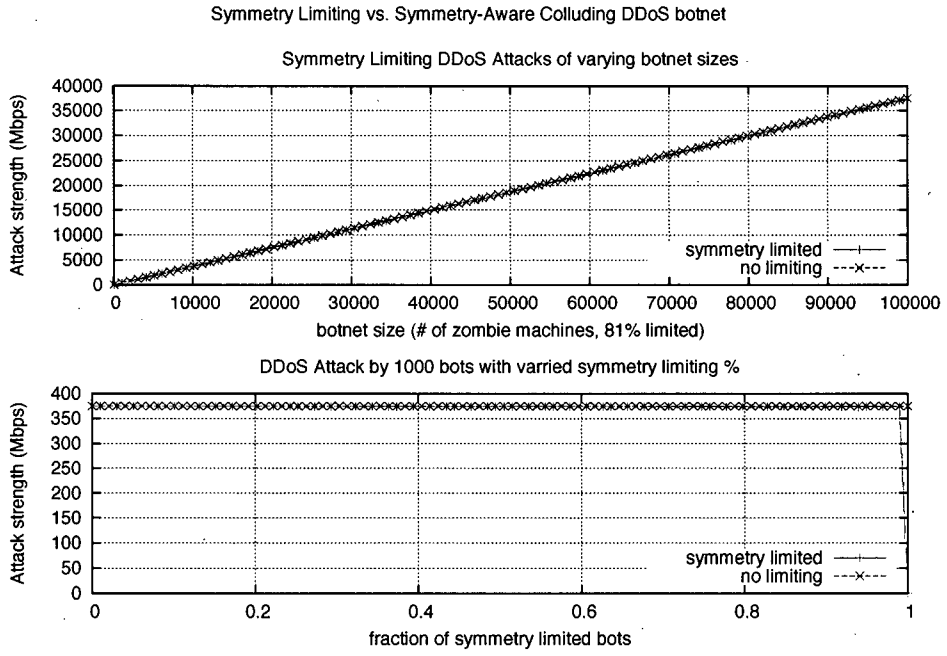


Figure 3.2: Symmetry-aware DoS flooding attack

$$\frac{L}{1-L} > c$$
$$L > \frac{c}{1+c}$$

Thus, the proportion of symmetry-limited bots (L) needed to reduce the attack strength significantly is dependent only on the values P_{attack}, P_{ack} and X. As shown in Figure 3.2, with $P_{attack} = 1500, P_{ack} = 40, X = 8$ yields $c = 300$, the corresponding L is 0.99667 – meaning limiting only becomes effective when nearly 99.667% of the bots are symmetry limited. The only parameter with any (real) freedom is the symmetry threshold X, as packet length standards are unlikely to change. Drastically tightening the symmetry limit to $X = 2$ – which would continue to permit the recommended TCP delayed acknowledgement implementation [3] – the symmetry limiter only comes into effect when $L = 0.9868$ or at 98.6% limiting coverage; still a very high percentage of the botnet.

While these numbers initially seem discouraging, the assumptions heavily favour the attacker, leading to over-estimation of the attack strength. The attack scenario assumes no packets are lost, the network delivers packets with zero latency, and the co-ordination between colluding zombies is precise – the timing of every forged ACK enables a symmetry-limited zombie to send a full X attack packets (ex. there is no interference by innocent use of the zombie machine by the PC's actual owner either). While the analysis could certainly model these factors to re-compute a more realistic attack strength, a much less tangible but far more significant factor in executing this attack from the attackers perspective, is the risk of exposure.

The need to forge back-traffic forces a botnet master to expose the network of machines under his or her control. Current botnets leverage the

anonymity of each bot when carrying out various computer crime. Any one bot need only communicate with the master IRCd server. If one bot is discovered, the identities of the other bots remain secret. However, the co-ordination to forge back-traffic across the entire botnet requires collaboration between bots, both in the discovery of which zombies can forge back-traffic, and which zombies need back-traffic forged for them, as well as during the attack. Traditional DoS defense techniques can capitalize on this wealth of colluding traffic to uncover the network of bots responsible for an attack, perhaps even leading to the dismantling of the botnet.

Additionally, there is the initial effort and the technical challenges to develop the collusion attack code. A symmetry-aware colluding DoS bot will require features to (i) determine which bots are symmetry limited, ingress filtered, or unfiltered, (ii) coordinate which unfiltered bots will forge back traffic for which symmetry limited bots, and (iii) synchronize the execution of the attack itself. This introduces significantly more control traffic into the botnet infrastructure, adding to the risk of exposing the botnet with such an increase in traffic exchanged between the attacking machines. Both the effort required to develop the code for this attack and the risk of exposure may guide the attacker by the law of diminishing returns, with the cost to execute such an attack greater than the value of the attack itself.

3.2.3 Limits of Symmetry Limiting

Packet symmetry enforcement adds an implicit signaling channel to the network, with P_B as the allowed admission rate to this channel. As the flooding attack analysis showed, P_B is linearly proportional to and is thus the limiting factor for flooding attack strength. Recalling that the P_B parameter must

allow a source to send at least one packet to a destination host to initiate (or bootstrap) communication that destination, it is clear P_B can never be reduced enough to defend all flooding attacks against victims of arbitrarily small provisioning. Tuning the parameters X and W in order to decrease P_B as a defense against pure volume-based botnet DoS attacks simply creates an arms race – as botnet sizes grow, P_B must shrink proportionally. Furthermore, shrinking P_B is difficult as X must be large enough to allow communication to proceed and W must be short enough not to significantly degrade the user experience. However, any connection-oriented or signaled network (ex. ATM, IP+RSVP, etc.) is attackable through denial-of-service on its control channel. Thus, as it is futile to choose parameter values based on attack strength, it remains most practical to define P_B on the basis of the characteristics and user acceptability of "good traffic".

3.3 Resilience to Attacks

The symmetry limiter is naturally resilient – being designed such that increasing attack power merely strengthens the defensive response. However, this section explores ways in which an attacker could attempt to attack the symmetry mechanism itself.

3.3.1 DoS your neighbour

An attacker cannot deny service to his or her fellow subscribers by spoofing source address information and corrupting their outgoing flows. A symmetry-limited network is assumed to ensure the integrity of source address information. Modern network equipment vendors (ex. [51]) ship products that make achieving network integrity a top priority, as source address

information is crucial for proper management of IP-based services. Thus, it is reasonable to assume source address information is not spoofable, which prevents an attacker from poisoning a neighbouring subscriber's symmetry.

3.3.2 Memory Exhaustion

An attacker could attempt to deny connectivity to other symmetry-limited subscribers by consuming all the available memory of the symmetry limiter. A symmetry limiter helps avoid such an attack as the per-flow state required to track packet symmetry is minimal. The memory requirements for a (simplified) symmetry tracking data structure breakdown as follows:

flow tuple...	
sip + dip (IPv6)	2 * 16 bytes = 32 bytes
prot	8 bits = 1 bytes
sport + dport	2 * 2 bytes = 4 bytes
window counters ...	
tx + rx	2 * 8 bytes = 16 bytes
window intervals ...	
tx + rx	2 * 10 * 4 bytes = 80 bytes
window timestamps ...	
tx + rx	2 * 4 bytes = 8 bytes
pointer to aggregate record ...	
parent	4 bytes
total	145 bytes / flow tracking record.

This minimal state allows a symmetry-limited network to support a large number of flows. For instance, with only 512 MB of RAM dedicated to symmetry tracking, a symmetry limiter can support $\frac{512MB*1024KB/MB*1024B/KB}{145B/rec} =$

3702558 flow records in total. In the worst case with all flows from completely different sources (i.e. none of the aggregate flow tracking records are shared by finer grained flow records) and each flow with 5 separate flow records, the limiter can track at minimum $3702558/5 = 740511$ flows. Under normal conditions, many flows will share flow tuples at coarser granularities, especially the 1-tuple for all flows coming from any one sender. A system administrator can thus configure the symmetry limiter according to the client-demand on the network, both in terms of the number of subscribers and the expected flows-per-subscriber.

The maximum flow capacity for the symmetry limiter extends the "good traffic" definition beyond just packets-per-flow to flows-per-client as well. An administrator may configure the system to allow 100 flows per client, allowing one limiter to support a network of nearly 7500 subscribers. Many clients will likely consume far fewer than 100 flows at a time, while other clients may at times exceed this threshold. Under normal operation, the limiter can remain agnostic to imbalances such as this. However, under conditions of extreme stress (i.e. memory exhaustion), discarding finer-grained records while retaining coarser-grained tracking records is made easy by the tree-structured organization of flow tuples. Furthermore, only those clients consuming more than their "fair share" of flows can be collapsed in this manner – punishing only the heavy consumers of the network with coarser grained tracking.

A clever attacker might attempt to exploit this memory conservation defense to execute a DoS attack. If the attacker could force tracking down to the 1-tuple `(sip,*,*,*,*)` granularity, he or she could establish a high rate 1:1 symmetric flow with another machine outside the symmetry limiter, which would enable the attacker to use the remaining $X - 1$ packets to

flood some other completely different destination IP address (since memory has been consumed to the point that only 1-tuple records are retained). To thwart such an attack, the limiter simply does not generalize flow tuples beyond the 2-tuple `(sip,dip,*,*,*)` granularity, ensuring that all destinations (i.e. potential victims) are tracked separately. This essentially places a cap on the maximum number of concurrent connections per client for a symmetry limited network. However, considering again only 512 MB of symmetry-tracking RAM, under extreme memory pressure, the limiter could support each of the 7500 clients with $f = 492$ flows to unique destination IP addresses (where f is calculated from $(1 + f) * 7500 = 3702558$), dropping any flows that exceed this limit.

Note the attacker would also require the ability to spoof source address information for this attack to be successful, since the memory collapsing mechanism ensures each sender only receives a fair-share of flow-tracking records at each granularity. As discussed above, it is assumed that source address information is unforgeable, making this attack impossible in the first place.

3.4 Summary

This chapter evaluated the effectiveness of DoS protection provided by symmetry limiting, assuming the parameter values for the symmetry threshold X and the window length W as established in Chapter 2. Both traditional bandwidth flooding and covert burst DoS attacks are severely limited with packet symmetry. Though symmetry-aware colluding DoS attacks could theoretically use forged back-traffic to maintain symmetry and execute powerful DoS attacks with symmetry-limited machines, the complexity to build the

attack code as well as the risk of exposing the precious botnet are strong deterrents against these collusion attacks. The minimal state used to track packet symmetry allows a symmetry-limited network to support a large number of flows while simultaneously resisting memory exhaustion attacks on the tracking mechanism itself.

Chapter 4

Implementing a Symmetry Limiter

A symmetry limiter prototype was built to evaluate the effects of symmetry limiting on the traffic dynamics on a live network. As per the ISP network deployment strategy, the prototype tracks and limits packets across a Linux network bridge. The values for the symmetry threshold X, the tracking window length W and the window interval length l are configurable, but the default values for the system are those derived from the traffic analysis ($X = 8, W = 10s, l = 1s$). The symmetry metric is computed as a flat value across the entire window, rather than the two-stage computation suggested to tolerate buggy TCP stacks in the analysis of asymmetric TCP flows. The implementation also does not implement any special flow tracking for multicast addresses, treating all IP addresses as the same.

4.1 System Architecture

The high-level algorithm used by the symmetry limiter is relatively simple. The system prevents the packet symmetry metric $\frac{\max(tx,1)}{\max(rx,1)}$ from Section 2.1.2 for any flow from exceeding the asymmetry threshold X. As such, for each IP packet received, the limiter (i) looks up the finest granularity flow record for the given packet (creating the records if they do not exist), (ii)

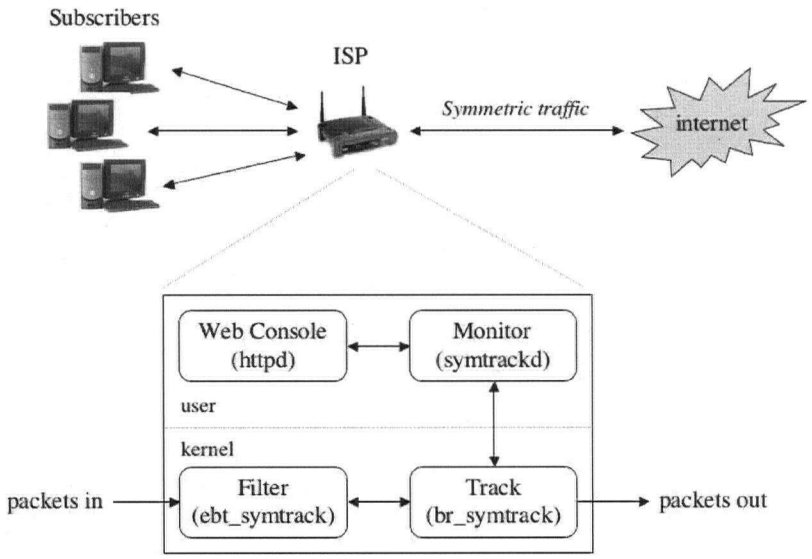


Figure 4.1: System architecture for Linux symmetry limiter prototype

checks if the packet causes the Tx:Rx ratio to exceed the $X : 1$ threshold, and if so, drops the packet, (iii) otherwise, updates the flow record counters and forwards the packet.

Figure 4.1 captures the overall system architecture for the symmetry limiter prototype. The architecture separates the tracking and limiting responsibilities for better encapsulation, making a clear division between parameters to tune symmetry tracking and those to tune symmetry limiting. A user-space daemon is responsible for pulling tracking and limiting data from the fast in-kernel data path, making this data available for display via the web interface, as well as potential further analysis.

The use of a Linux network bridge allows a limiter to be transparently

deployed on any link joining two (or more) networks. Neither end hosts nor routing infrastructure require any configuration changes – the limiter simply *bridges* two networks, and any limited traffic just looks like any other network loss.

Performance was a key factor in the design. The system must achieve high throughput for source-based symmetry limiting to be practical. A corner stone of the argument in favour of packet symmetry is that symmetry limiting does not negatively affect innocent users. However, a significant degradation in throughput or performance would have a negative on all users – innocent and malicious alike. As such, careful attention was paid to optimizing the data path that packets traverse through the symmetry limiter. The individual performance considerations for each module are discussed in the sections below.

4.1.1 Tracking Packets

The packet symmetry tracking module (br_symtrack) uses the Netfilter ethernet bridge to hook into the arrival of packets traveling across the bridge. This Netfilter ethernet bridge is built into 2.6 Linux kernels, and can be patched into earlier kernel versions. The tracking module can be loaded dynamically into (and out of) the kernel, however, flow tracking data is not persistent across loads and unloads. This module consists of 1767 lines of documented C code and took three months to develop.

Flows are tracked at all flow-tuple granularities, from the full 5-tuple to the source-only 1-tuple. The flow tuples at each of the 4 aggregate granularities are shared by all of the flows at that granularity. Thus, flow tuples are organized in a tree structure, branching out at each successive granu-

larity. The tree is bidirectional – child flow tuples have a pointer to their parent, and a parent has a linked-list of pointers to all its children. A single global record tracks the packet symmetry of the entire network, which could be considered the 0-tuple granularity. All 1-tuples have this global record as their direct parent, making the 0-tuple record the root of the tree of all tracking records.

Flow tracking occurs from finest-to-coarsest granularity. For a given packet, the tracking module first looks for the corresponding 5-tuple. If that is not found, then the 4-tuple is searched for, and so forth, until the 0-tuple global record is reached. When appropriate tuple is found (note that we will always find the 0-tuple), all missing finer-granularity flow-tuples above the one found are created, up to the full 5-tuple granularity (or a configurable finest-granularity setting). Any new records have the proper parent-child links set such that the flow tuple tree remains consistent. Then, the (tx, rx) counters are updated to reflect this new packet, traversing the finest to coarsest tuple for the packet. The window history is updated for each flow tuple, using the arrival time of the new packet as the current time. This process optimizes the data path by maximizing the work done for the first packet seen in a flow, and minimizing the per-packet maintenance to continue tracking the flow. Once the first packet has been seen, all flow-tuple granularities have been created for that flow, and each subsequent packet merely traverses its branch in the tree from finest-to-coarsest granularity, updating the windowed counters along the way.

A single hashtable stores all the flow-tuples, using chaining to resolve collisions. The size of the hashtable defaults to use 1/16384 of memory, as suggested by the Linux *tcp.c* code, but this can be configured at compile time. A tracking record at any granularity is expired and purged, having

its memory reclaimed, after 30 seconds of inactivity (though this expiration time is configurable and can be changed dynamically).

A single global read-write spin-lock is responsible handling concurrent access to flow tracking records. While this choice favours the assurance of correctness over performance for the tracking module, other components in the architecture take this into account to control any potential negative effects on performance.

Flow tracking data is exported using the */proc* pseudo file system, with separate files for each flow-tuple granularity, and one file for all granularities.

files in /proc/br_symtrack/...	
counters_src	1-tuple counter records
counters_src_dst	2-tuple counter records
counters_src_dst_prot	3-tuple counter records
counters_src_dst_prot_dport	4-tuple counter records
counters_src_dst_prot_dport_sport	5-tuple counter records
counters_all	all counter records

To optimize the data path, the *seq_file* subsystem is used to export data to */proc*. This subsystem sets up data access as a pull operation – data moves from kernel to user space only when data is requested (i.e. via a file read on one of the above */proc* files). The *seq_file* API presents an iterator interface for the kernel module on which to map the data to be exported. The global spin-lock is held with a write-lock while this iterator traverses the hashtable of flow tuples. This allows the traversal to purge stale tracking records while active flow records are written as output. Again, although this locking strategy favours correctness over performance, other components

are sensitive to this choice and coordinate their actions to prevent possible performance issues.

The tracking module provides a public API for the limiting module to lookup and query flow records with respect to their packet symmetry. The flow tracking structure also contains extra windowed counters to store the packet drops for that granularity. The tracking module exports a function to update these counters, thus tightly coupling packet dropping data to the flow records, while remaining orthogonal to the actual packet drop enforcement module. As packet dropping counters are stored in the tracking module, the module also pushes changes to flow limiting status to user space via another character device, */dev/symtrack_livelog*. The tracking module exports the flow-tuple along with the starting and ending times when the flow tuple experienced limiting. The tracking module assumes that limiting starts when the flow's packet drop counters are incremented above 0 for the first time, and that limiting ends when these drop counters return to 0 again.

The flow tracking data structure consumes a total of 320 bytes. This is larger than the size suggested in Section 3.3.2 as extra memory is consumed with the fields to store the record in the hashtable, the linked-list pointers for parent-to-child tree traversal, the extra packet drop windowed counters, the limiting state bits, and padding between each field. Though this makes the actual record size a little over twice that of the predicted tracking structure, this level of memory usage still scales to support a large number of flows. Considering the worst case with completely independent flows (i.e. no sharing of aggregate flow-tuple records), with 512 MB RAM, this tracking module prototype can support 512 * 1024 * 1024 bytes / 320 bytes per records = 1,677,721 flow records, which supports a minimum of 1,677,721/5 = 335,544 total flows. Estimating the average number of flows-

per-client at 100, such a limiter can still support over 3300 clients. This result is encouraging for the scalability of symmetry limiting, suggesting a special-purpose symmetry-limiter network appliance may scale to the needs of modern ISPs.

4.1.2 Filtering Packets

The filtering module (ebt_symtrack) is an extension module for the ebtables ethernet bridge firewall. This module uses the public API of the tracking module to query for flow tracking data, consequently dropping packets for flows that exceed a symmetry threshold. The symmetry threshold is a dynamically configurable value possessed by the filtering module. This module consists of 194 lines of documented C code and was developed in tandem with the tracking module (br_symtrack).

Ebtables is essentially the equivalent of the iptables firewall, only at the ethernet layer. It provides three chains, or *tables*, for ethernet frame arrival; INPUT for frames destined to the machine, OUTPUT for frames sent from the machine, and FORWARD for frames passing through the machine (i.e. bridged frames). The FORWARD chain is the one used by the filtering module. Rules can be added to each of these chains, where each rule examines properties of the frame (i.e. MAC or IP addresses), and based on these properties the frame can be accepted or dropped. The filtering module adds a new rule type to ebtables that accepts or drops frames based on the symmetry for the flow to which the frame corresponds. This symmetry limiting rule provides options to configure the symmetry threshold X, as well as the flow-tuple granularity at which limiting is to occur. By default, the limiter uses the finest-granularity limiting strategy

discussed earlier, but allows a specific flow-tuple granularity to be chosen. There is no default value for the limiting threshold X, forcing the calling system to specify this value explicitly.

The filtering module is relatively simple compared to the tracking module. The symmetry limiting rule implementation simply uses the per-packet hook of the ebtables rule framework to decide whether to accept or drop a given packet. In this hook, the module merely calls into the tracking module to lookup the tracking record for the flow that corresponds to the packet in question (which is provided as an argument to the hook callback). If the flow's symmetry exceeds the threshold specified by the rule, the rule marks the packet to be dropped, otherwise the packet is marked as being accepted by the symmetry rule, and goes on for further processing by any remaining rules in the FORWARD chain. The filtering module also uses the public API to increment the packet drop counters for flows that are subjected to symmetry limiting, providing valuable feedback to the tracking module regarding limiting behaviour.

4.1.3 User-level Daemon

The user-level daemon process acts as the control centre for the symmetry limiting architecture. It consists of 1783 lines of documented Python code and was developed in less than one month. This process sets the ebtables rules to limit flows based on symmetry, specifying at which threshold and granularity to limit. The daemon is also responsible for marshaling the tracking and limiting data from kernel space to user space, reading directly from the */proc/br_symtrack/counters...* files and from */dev/symtrack_livelog*. The interval at which the kernel data is read is configured on startup, and

defaults to poll once every 2 seconds. At present, the kernel data is simply copied to user space and stored in the file system for use by the other system components.

It is important that the polling interval used to fetch tracking data from the kernel is not too short, given the locking strategy employed by the kernel tracking module. Extremely frequent reads to the */proc/br_symtrack/counters...* files could cause a severe performance degrade, recalling that the kernel tracking module holds an exclusive write-lock while writing tracking data output. Thus, centralizing the access to the kernel data with this daemon process prevents an attacker from exploiting this implementation decision. The system administrator controls this polling interval, allowing it to be configured to scale as needed. This default value of 2 seconds was chosen rather arbitrarily to provide instantaneous-enough access to tracking data. However, this value has seemed to have little negative effect on overall system performance.

An additional mechanism built into the system purely for the sake of encouraging deployment is the ability to exempt a specific flow granularity from symmetry limiting. This feature is implemented by the daemon process, by inserting an additional ebtables rule per exempt flow just ahead of the symmetry limiting rule in the FORWARD chain. Each of the flow exemption rules simply check that the IP address and port number information matches those of the exempt flow granularity, and accept the packet, stopping the traversal of the remaining ebtables rules. To set new exemptions, the daemon checks certain drop-box files, at the same polling interval used to read the tracking data, for requests to exempt specific flows from symmetry limiting. These requests are simply ascii text detailing the flow-tuple to be exempt and a time delta of how long the exemption should remain

active. Thus, the daemon also regularly checks all flow exemptions for any that have expired, promptly removing that flow's exemption rule from the ebtables chain.

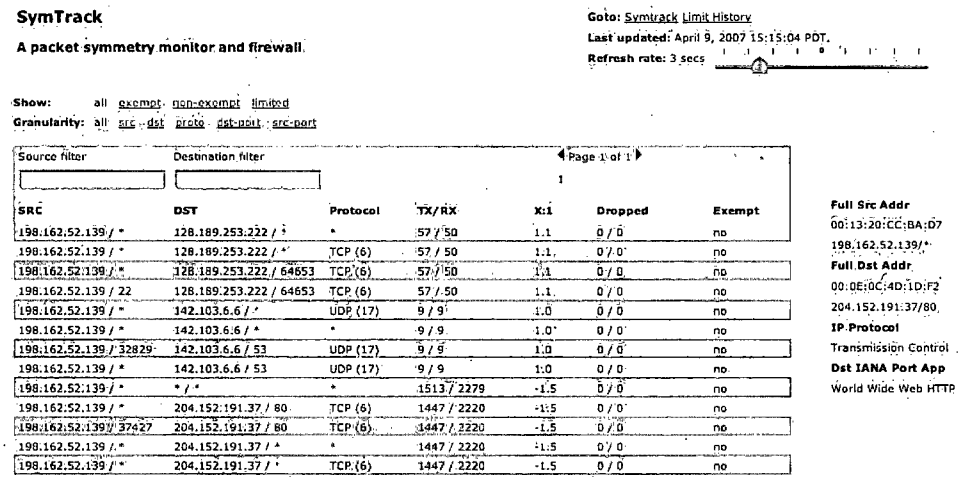
4.1.4 Web Console

The main purpose of the web interface was to make symmetry tracking data accessible to the end user. This dynamic AJAX front end provides access to the flow tracking data in real-time, with a configurable refresh rate, as shown in Figure 4.2. For privacy, a symmetry limited client is only given the flow tracking data for its own flows. Only a system administrator connecting directly to the symmetry limiter machine is able to view the tracking data for all flows traversing the bridge. The web interface consists of 1731 lines of HTML and Javascript code and was developed over the course of two months.

The interface also allows end-users to request limiting exemptions for flows that may be exceeding the symmetry threshold. Recall again this mechanism is only incorporated to encourage and facilitate experimental deployment, since an end-user would always have the ability to turn off the mechanism at any time. Such an exemption mechanism available to all clients should not be incorporated into a commercial deployment, as attackers could simply turn off limiting for their attacks.

4.2 Performance Evaluation

The Emulab system [79] was used to create a network topology to build and evaluate the symmetry limiter prototype. The experimental network topology is given in Figure 4.3. Machine B was configured as a bridge



SymTrack

A packet symmetry monitor and firewall.

Goto: Symtrack Limit History
Last updated: April 9, 2007 15:15:04 PDT.
Refresh rate: 3 secs

Show: all exempt non-exempt limited
Granularity: all src dst proto dst-port src-port

Source filter | Destination filter | Page 1 of 1

SRC	DST	Protocol	TX/RX	X:1	Dropped	Exempt
198.162.52.139 / *	128.189.253.222 / *	*	57 / 50	1.1	0 / 0	no
198.162.52.139 / *	128.189.253.222 / *	TCP (6)	57 / 50	1.1	0 / 0	no
198.162.52.139 / *	128.189.253.222 / 64653	TCP (6)	57 / 50	1.1	0 / 0	no
198.162.52.139 / 22	128.189.253.222 / 64653	TCP (6)	57 / 50	1.1	0 / 0	no
198.162.52.139 / *	142.103.6.6 / *	UDP (17)	9 / 9	1.0	0 / 0	no
198.162.52.139 / *	142.103.6.6 / *	*	9 / 9	1.0	0 / 0	no
198.162.52.139 / 32829	142.103.6.6 / 53	UDP (17)	9 / 9	1.0	0 / 0	no
198.162.52.139 / *	142.103.6.6 / 53	UDP (17)	9 / 9	1.0	0 / 0	no
198.162.52.139 / *	* / *	*	1513 / 2279	-1.5	0 / 0	no
198.162.52.139 / *	204.152.191.37 / 80	TCP (6)	1447 / 2220	-1.5	0 / 0	no
198.162.52.139 / 37427	204.152.191.37 / 80	TCP (6)	1447 / 2220	-1.5	0 / 0	no
198.162.52.139 / *	204.152.191.37 / *	*	1447 / 2220	-1.5	0 / 0	no
198.162.52.139 / *	204.152.191.37 / *	TCP (6)	1447 / 2220	-1.5	0 / 0	no

Full Src Addr
00:13:20:CC:BA:D7
198.162.52.139/*
Full Dst Addr
00:0E:0C:4D:1D:F2
204.152.191.37/80
IP Protocol
Transmission Control
Dst IANA Port App
World Wide Web HTTP

Figure 4.2: Screen shot of symmetry-limiter web interface.

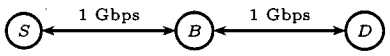


Figure 4.3: Symmetry limiter evaluation network topology.

between host S and D. All machines were equipped with Pentium IV 3200 MHz processors, 1024 MB of RAM, and ran a standard image of Fedora Core 4. The machines were connected on a VLAN-switched network with 1 Gbps links.

The following experiment determines the performance impact of the symmetry limiting prototype on network throughput. As the prototype targets deployment on a Linux bridge, the maximum throughput of the system is confined to the maximum throughput of the Linux bridging architecture. As

such, the evaluation methodology first measures the throughput of the raw Linux bridge. This forms the baseline to which symmetry limiting throughput is compared.

The *ttcp* benchmark was used to measure the TCP throughput achieved between S and D. The baseline configuration uses the raw ethernet bridge setup as described above. The symmetry limiting configuration uses the full prototype architecture of symmetry tracking and limiting modules, as well as the daemon process with the default polling rate of 2 seconds. Four separate runs measure the effect of junk flooding traffic on throughput performance, both for the baseline and symmetry limiting configurations. The data points measured 0, 4000, 8000, and 16000 packets per second of noise traffic. The noise traffic consisted of packets with randomly generated IP and layer 4 header information to increase the workload of the symmetry limiter to create records and track these additional flows. A single trial consisted of sending 262144 packets, which took roughly 25 seconds for each trial. Five trials were conducted for each run, and the average throughputs are reported in Figure 4.4. Symmetry limiting throughput degrades equally with the throughput of the raw Linux bridge, independent of extraneous noise traffic, which one would expect to slow symmetry tracking to some degree. As such, this result is promising and suggests that symmetry limiting imposes a small and constant per-packet overhead.

Also of interest, *oprofile* results indicate that the limiter spends the majority of its CPU cycles in two functions of the tracking module; the flow tuple lookup function used to find a tracking record in the hashtable, and the packet parsing function used to pull the flow-tuple fields out of the raw packet data. This suggests that the single-lock strategy combined with the 2 second polling rate does not impact performance significantly – otherwise

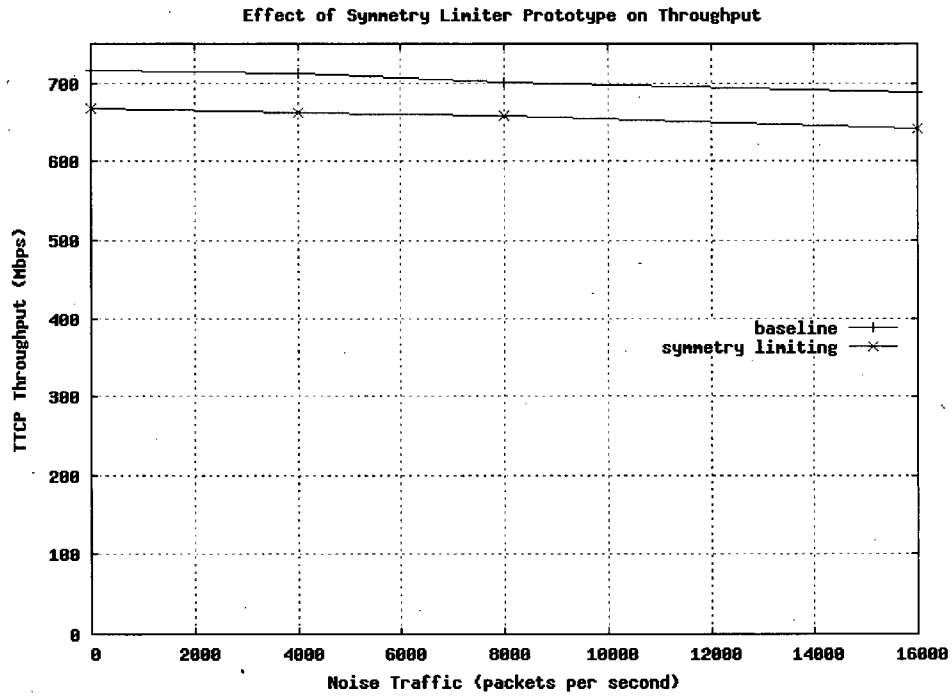


Figure 4.4: Comparing symmetry limiting throughput performance.

one would expect significant time spent in the iterator routine used to write tracking data to user space.

4.2.1 Future Optimizations

Although the performance of the prototype is rather encouraging, there are a number of optimizations that would improve performance to an even greater extent. As suggested by the *oprofile* results, the largest gains could be made by improving the hash lookup and packet parsing code.

The time spent in the hash lookup function could be reduced drastically (nearly 50%) by introducing tighter coupling between the tracking and limiting modules. Currently, the same hash lookup occurs twice for every packet that is not dropped due to symmetry limiting – the first time in the limiting module to verify the flow is not over symmetry, and the second time in the tracking module to update the flow tracking structures. While this embodies good software design principles of encapsulation and separation-of-duty, it is also slower. An optimization might use another field in the packet's socket buffer structure (which is passed to both the tracking and limiting callbacks), to store the pointer to the symmetry tracking record for that packet. The limiting module, having done the initial hash lookup, could set this pointer and the tracking module could check this field before doing its own lookup. This would optimize the common-case, as for all packets but the very first packet for a flow, the tracking record should be found by the limiting module and this pointer-sharing should succeed.

The packet parsing code could also be optimized by tighter coupling. Currently, the full 5-tuple of fields are always parsed from the raw packet data, after which the mask of the desired flow-tuple granularity is generi-

cally applied. This could be trivially optimized by taking the granularity mask into account while parsing the raw packet fields, again likely reducing the cycles spent in this function. Notice that the packet parsing also occurs twice per-packet, which could be reduced similar to the hash lookup function by sharing results across the tracking and limiting modules. However, independent parsing of the packet buffer would allow the tracking module to verify the tracking record in the packet buffer actually corresponds to the correct flow.

Although the performance measurements do not indicate the single-lock strategy as a bottleneck, additional improvements could certainly be made there as well. Firstly, with sufficient memory space available, purging stale records is perhaps not as important as high flow throughput. Thus, if the iterator to output tracking data is changed to no longer purge stale records, that process would only require a read-lock, which can be held while existing flows continue to pass through the limiter. This would still prevent new flows from being created while the output was being written since creation (like deletion) requires a global write lock to the hashtable to prevent parent records from being deleted while new children are created. A three-lock hierarchy might solve this problem, where only the first lock is needed to read the hashtable, the first and second locks are needed to create new flow records, and all three locks are required to delete records.

To further optimize the data exchange between kernel and user space, the */proc* pseudo-files could export binary data rather than ascii character data. Simply writing the raw bytes of the flow tracking structure would save CPU time consumed by the string formatting routine currently used to produce a relatively human-readable representation of the data. However, as with the previous optimization, the kernel-to-user space data transfer has

yet to become a system bottleneck.

4.3 Deployment Experience

To evaluate symmetry limiting against real traffic, the prototype was deployed on a network bridge between the UBC Distributed Systems Group (DSG) lab network and the external internet. The bridge machine ran an unmodified Linux 2.6.16 kernel on a 2.26GHz Intel P4 processor with 631 MB of RAM. The symmetry tracking and limiting modules were run on this machine, along with the web server to provide the symmetry-limited machines access to the live tracking and limiting data for their flows.

Live deployment affords the analysis of symmetry limiting effects on live traffic dynamics, which is important as such effects cannot be simulated in the analysis of network traces. A secondary aim is to discover protocols or applications with asymmetric behaviour that were not represented in the network traces examined. Consequently, the focus of deployment is more to evaluate any usability effects of symmetry limiting, i.e. determining if any protocols or applications actually break subject to symmetry limiting dynamics. Catching real attack traffic is of much less concern, as the user base of computer systems researchers are unlikely to mount real attacks. As the results show, symmetry limiting had little to no effect on end-user experience, even in light of some traffic being limited.

4.3.1 A Symmetric Observation

An original concern was that voice-over-IP (VoIP) traffic may exhibit significant asymmetry. The nature of VoIP applications – high real-time demands to deliver voice data immediately combined with mild tolerance for data

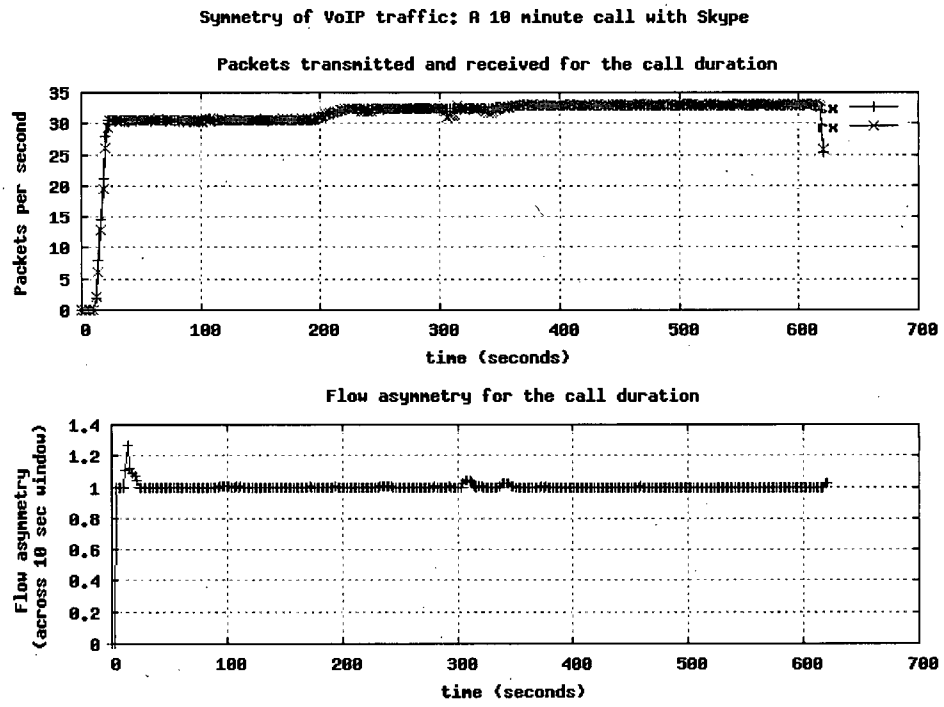


Figure 4.5: Symmetry of Skype VoIP traffic. Although several periods of one-way conversation occurred during the call, tx:rx packet symmetry remains steady around 1:1 for the entire call duration.

loss – lead most VoIP implementations to use UDP as an underlying transport protocol, avoiding possible transmission delay due to TCP congestion control algorithms. However, experience with Skype [70], a popular (free) multi-platform VoIP application which uses UDP as underlying transport, showed that VoIP is highly symmetric, maintaining a nearly 1:1 Tx:Rx ratio throughout entire calls. Figure 4.5 illustrates the symmetry exhibited by Skype's VoIP traffic for one call placed through a DSG symmetry-limited machine. This near-perfect packet symmetry is suggestive of an 'always-on' mode for VoIP operation – that a VoIP client need not distinguish between on or off periods in which to send or not send sound traffic. Rather, it simply sends all sound traffic from the source end, considering ambient background noise equally valid as part of the conversation.

4.3.2 Asymmetric Observations

Multicast DNS

Traffic to UDP port 5353 was frequently limited – between zero and 5 times per day. Port 5353 is the standard port for multicast DNS (mDNS) [23], an extension to DNS to perform queries over IP Multicast to provide zero-configuration transparent connectivity to a variety of devices on the local link. The protocol has been designed with multiplicatively increasing intervals between queries as well as a need-to-know query policy to minimize bandwidth and CPU consumption for smaller less powerful devices (i.e. cell phones, PDAs, etc.). However, when network connectivity for a device changes (i.e. connects to network, wakes from sleep, etc.), it must perform the mDNS start up phase. During start up, the machine first must probe the local link for any other machines with conflicting names as the

machine starting up. Probes may contain multiple DNS questions in a single packet, and thus may only require a single packet (we observed only single packet probes). Each probe request must be separated by 250 ms intervals. If no conflicting responses are received, the machine then moves to the announcing phase, during which the responder must send at least 2 responses containing all the machine's link-local resource records. The responses have multiplicatively increasing delays, starting from 1 second and doubling for each subsequent response.

For the DSG deployment, symmetry limiting only occurred during start up of a responder. The start up phase is the most asymmetric portion of the protocol, as probing with no conflicts followed by announcing leads to a short but high rate datagram stream from the responder to the multicast address, with no back traffic to signal to the symmetry limiter that this is desired. However, this presents little concern to using symmetry as a DoS defense mechanism. Firstly, the bootstrapping packet rate allows a one-packet resource record set to complete a no-conflict start up phase without experiencing any limiting (3 probe packets and 2 response packets). From the DSG traces, no mDNS packet seen was more than 350 bytes and mDNS allows up to MTU sized datagrams as an extension, indicating there is significant room for larger resource record sets that fit within a single packet. Thus, without any changes to symmetry limiting, the basic usage of mDNS is supported.

The limiting that actually occurred, however, seems to be caused by buggy (or misbehaving) mDNS clients. During what appears to be the start-up phase of mDNS, the client sent unsolicited mDNS responses (responses to no queries) as well as disobeyed the recommendation (an RFC "SHOULD") to group queries and responses into the smallest number of network packets

possible. As such, the client essentially performed two simultaneous start-up phases, probing for different resource record sets in each case, rather than simply using one start-up phase to probe for all records. Although the prototype does not support special symmetry tracking for multicast addresses, including such special tracking would not prevent the flows described above from being limited. No back-traffic from other mDNS responders on the local link was observed. As such, the flows would remain equally asymmetric and be subjected to the same limiting.

TCP Asymmetry

Several TCP flows experienced symmetry limiting for the use of large congestion windows after a relatively long period of inactivity cause several bursts of asymmetric TCP traffic. As discussed earlier in Section 2.4.3, this problem is a known implementation bug in TCP stacks [3], which manifests particularly with the re-use of idle TCP connections for persistent HTTP transfers.

Another cause of asymmetric TCP traffic that was not represented in the trace analysis is the opening and/or closing of several TCP connections in parallel. Several peer-to-peer file sharing or content distribution applications demonstrate this behaviour on start up, shutdown or when new content is requested from multiple peers. The actual observed limiting was caused by an RSS feed reader application closing all its outstanding TCP connections, presumably on shutdown. This occurred after an extended period of inactivity, perhaps while the end-user read the available feeds. After this idle period, the sliding window history was lost. While the first X TCP FINs got through the limiter unscathed, these X FIN packets push the 1-tuple

granularity's asymmetry to the $X : 1$ threshold, and due to the limiting algorithm's traversal of all aggregate flow records for 5-tuples with $rx = 0$, subsequent FIN packets were dropped.

Thus, the symmetry threshold also places an upper bound on connection parallelism, preventing the number of outstanding TCP SYN or FIN requests from exceeding X. This is reminiscent of Microsoft's limit on half-open TCP connections [5] which was to the great discontent of Windows-based peer-to-peer application users, prompting the users to find means to subvert the throttling mechanism [57]. This suggests users would be less than tolerant of their peer-to-peer applications malfunctioning due to symmetry limiting. However, such limiting could be prevented by only limiting at 2-tuple and finer granularities. While this prevents symmetry limiting from actively blocking host scanning activity, this represents a tradeoff for a system administrator to balance.

4.3.3 Security Evaluation

Having a symmetry limiter prototype deployed, it seemed prudent to demonstrate the effectiveness of symmetry limiting against a DoS flooding attack. Rather than using genuine DoS attack tools from the wild, this experiment uses the same *ttcp* program to create the innocent flow and the malicious DoS flood that attempts to block the innocent flow from completing. The network topology used for this experiment is given in Figure 4.6, with each machine connected on a switched VLAN with 100 Mbps links.

Figure 4.7 demonstrates the effectiveness of symmetry limiting against a UDP DoS flood. The scenario is the same for both plots in the figure; the innocent *ttcp* transfer begins at time zero, and after 10 seconds a UDP

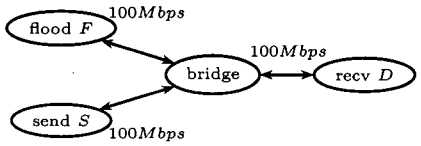


Figure 4.6: Network topology for example DoS flood attack.

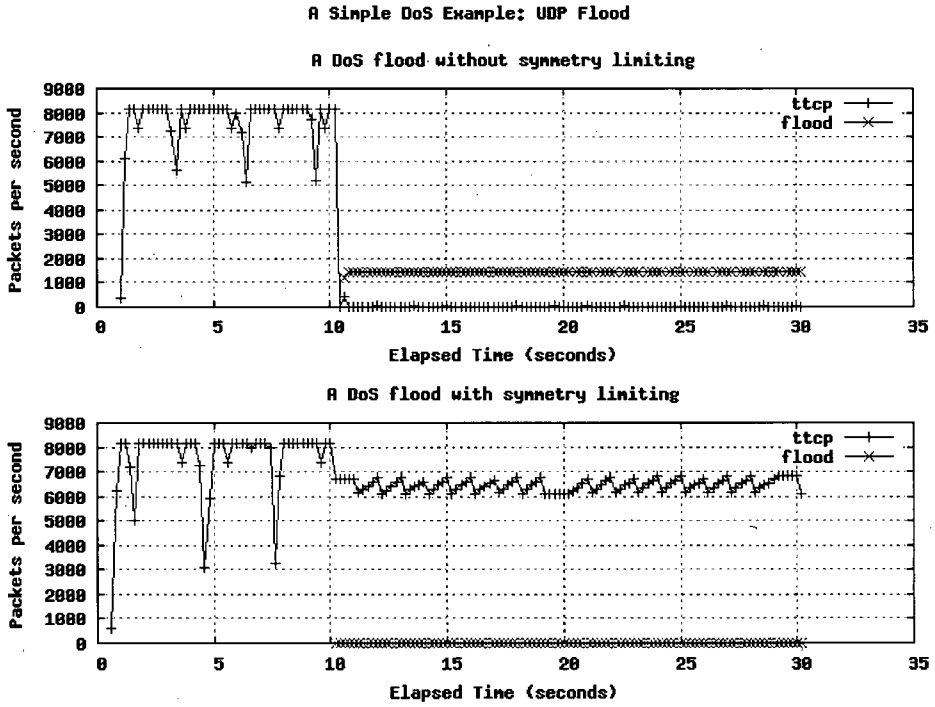


Figure 4.7: Effectiveness of symmetry limiting vs. a UDP flood.

flood is started. In the first plot without symmetry limiting enabled, the UDP flood completely saturates the link through the bridge and prevents the innocent transfer from making any progress. Note that the UDP flood packet rate is much lower that the TCP transfer since the UDP flood uses larger 8192 byte packets while the TCP transfer uses standard 1500 byte packets. In the second plot with symmetry limiting enabled, although the UDP flood slightly hinders the data transfer's performance slightly, this innocent flow maintains a high throughput as the UDP flood is throttled down to essentially $P_B = 1$ packet per second.

4.4 Summary

This chapter discusses the implementation and deployment of a symmetry limiter prototype. The prototype tracks and limits flows across a Linux bridge and provides a web interface to view the tracking and limiting data. The performance evaluation suggests the symmetry limiting mechanism imposes a small and constant per-packet overhead on the network, which builds confidence in the scalability of in-network symmetry limiter deployment. The live deployment of the prototype on the link connecting the UBC Distributed Systems Group lab network to the external Internet illustrated the minimal and largely innocuous effect of symmetry limiting on application level networking. Lastly, the prototype demonstrated the effectiveness of symmetry limiting against an example DoS attack, maintaining high-throughput for an innocent symmetric flow in the face of a malicious bandwidth flood.

Chapter 5

Related Work

5.1 Identifying Sources

Ingress filtering [31] pioneered the use of a good-traffic definition for source-based filtering by only allowing an end host to send packets with a genuine source address (an address that starts with a prefix that is advertised by the network). Although the initial acceptance and deployment of ingress filtering was low, the Spoofer project [15] currently estimates that less than twenty percent of net blocks are spoofable, indicating that ingress filtering is widely deployed today. However, this provides aggregate source address authentication – a source behind the ingress port can still masquerade as another host on the same network. Precise binding of an IP address to a machine requires the end host have a dedicated physical link that is not shared with other hosts, as in a switched LAN. Access servers, where a PC connects to a server using a login-password or other credential, to gain access to the network can also provide finer-grained control over source address binding. As discussed, symmetry limiting requires fine-grained source address integrity to prevent malicious hosts from denying service to other innocent hosts on the same network.

Traceback [12] and packet marking schemes [65] reconstruct reverse paths from victim to attacker(s), assuming IP address information can (and will)

be spoofed. However, the probabilistic nature of these schemes requires a large volume of traffic to be observed before enough path information can back propagate to each hop along the path. This impedes victims of smaller scale attacks from using these techniques, and the period of observation delays the investigation and response to large scale attacks.

5.2 DoS Attack Detection

Change-point monitoring [77] uses stateless, flow-independent statistics to detect DoS activity. The solution tracks TCP SYN/FIN and SYN/SYN-ACK pairs, which are inherently unbalanced for the most common DoS attack, the TCP SYN flood. Like packet symmetry, change-point monitoring is also effective when deployed at the source network, since disproportionate SYN-ACKs will return to the attacker's network. However, this solution is tightly coupled to TCP protocol semantics and specifically detects SYN-floods only. Attackers can remain undetected by simply launching UDP-based attacks. On the other hand, symmetry limiting is agnostic to protocol and can thwart flooding attacks regardless of the IP protocol used.

The MULTOPS data structure [35] is designed to detect and combat bandwidth DoS attacks agnostic to protocol, using the balance of incoming and outgoing packet rates. MULTOPS is a tree that tracks packet rates for aggregate IPv4 address blocks at byte-level granularity. The tree expands and contracts within a confined memory space, moving from coarser to finer prefixes as packet rates for those prefixes increase. This strategy is meant to thwart attacks against the MULTOPS structure itself. However, initially tracking at coarser granularity allows attackers to hide malicious traffic within a larger body of normal-looking traffic. As such, symmetry

limiting only moves from finer to coarser tracking during periods of extreme stress. The finer per-flow symmetry limiting prevents attackers from hiding attack flows within a larger body of relatively balanced flows, as each flow is tracked separately and is subject to independent limiting.

5.3 Reactive DoS Defense

Reactive filtering solutions typically operate in two stages; first detecting an attack and then taking action (filtering) to combat the attack. Symmetry limiting already has an advantage over these solutions in being proactive – symmetry limiting operates in an always-on mode without any threshold to detect attacks that trigger countermeasures. Furthermore, most of the solutions discussed below suffer from the need for a large initial deployment in order to be effective. While (for the most part) technically sound, the solutions are not economically viable due to the critical mass of source- and destination-end hosts or routers needed to have a significant impact on DoS attacks today. Some techniques are further hampered with requirements for ongoing collaboration of nodes that reside in separate administrative boundaries – adding much to the initial cost of deployment. Symmetry limiting has the benefit of providing immediate benefit to a deploying ISP, with each and every independent deployment increasing the strength and value of the network.

5.3.1 Victim-Based

Hop-count filtering [43] is a victim-based DoS solution that filters packets with spoofed source IP addresses on the assumption the spoofed packet will have an incorrect hop-count value. The hop-count can be effectively deduced

from the TTL value observed at the destination under the assumptions that (i) Internet hosts are at most 30 hops apart and (ii) various operating systems use customary initial TTL values. However, this scheme requires an initial phase to build a table containing the correct hop-count values for the IP addresses of desired clients (not to mention a process to identify those desired clients). As well, this table remains a challenge to maintain throughout the lifetime of the service. Those issues aside, widespread deployment of this solution simply creates an arms race for attackers to devise attack code to automatically generate the correct hop-count values of desired clients. As well, [53] shows that the majority of DDoS attacks do not make use of IP spoofing.

5.3.2 Router-Based

ACC-Pushback [40, 50] has each router locally monitor the ambient packet drop rate of each link coming into the router. A link is determined to be under attack if the drop rate exceeds a pre-defined threshold (the default is 10%). The router locally calculates a rate limit for each aggregate flow (i.e. IP address prefix) to bring said ambient drop rate back within the acceptable range. The rate limit is then pushed upstream to other ACC-Pushback routers closer to the source of the attack traffic. Unfortunately, innocent traffic falling in the same aggregate as malicious traffic is also rate-limited, possibly resulting in significant collateral damage. As well, ACC routers authenticate each other based on a TTL value of 255. Thus, Pushback must be deployed on direct neighbouring routers for upstream filtering to occur, which presents economic and political challenges for widespread deployment.

Max-min server-centric throttles [82] is a simplified version of Pushback

where no aggregate flow is specified, but all the traffic through a router destined for a particular host is limited to a particular (fair) rate. When the destination is under stress, the rate is pushed out k-hops from the destination towards source-hosts so malicious traffic is limited. The key insight is the distributed computation of k that maximizes the limiting of attack traffic and minimizes the limiting of innocent traffic. Though, regardless of how minimal, innocent traffic behind the k^{th} hop becomes collateral damage. Deployment is also hindered by the complex control feedback loops required to monitor and compute k, loops which must cross administrative boundaries. As well, the system does not address widely distributed attacks in which different attack aggregates have a different optimal values of k.

Active Internet Traffic Filtering [8] is another router-based solution, but avoids collateral damage by tracking per-flow behaviour. AITF targets deployment at the network edge, preferably at ISP edge routers. Each AITF router appends their IP address and a random nonce into a section between the IP header and TCP header. The chain of IP addresses form the path needed by an AITF router at the victim end to contact the appropriate AITF router at the source end of the attack to request filtering. The authors admit the (gratuitous) use of random numbers as an authentication mechanism allows malicious hosts to forge messages and hinder the effectiveness of AITF. And again, AITF requires deployment at both source and destination ends to be effective, adding the challenge of building political momentum and economic backing for the solution to the deployment strategy.

Congestion puzzles [78] are another router-based defense mechanism, in which the client must solve cryptographic puzzles at a rate equal to the desired sending data rate. In theory, this currency-based approach forces a client to pay for consumed bandwidth with its own computing power.

However, an attacker can coax a large number of otherwise innocent clients into solving many puzzles on the attacker's behalf, allowing the attacker to gain an unreasonably large flow rate. Unlike the previous solutions, this scheme requires widespread deployment in both routers *and* end hosts, as end hosts must implement a congestion-puzzle-aware protocol directly in the OS network stack.

D-WARD [54] proposes source-based DoS filtering using per-flow outbound and inbound packet rates to detect attacks. While TCP is subject to balanced outbound and inbound packet rates, ICMP and UDP are subject simply to maximum rates since acknowledgements are not explicitly built into these protocols. As such, the mechanism is vulnerable to low-rate UDP attacks that simply consume the maximum rate. Flows are also considered guilty-before-proven-innocent, potentially resulting in innocent clients being denied service while the network suffers from even mild congestion. Furthermore, the router-based deployment assumes IP spoofing is possible, leaving the mechanism vulnerable to memory exhaustion attacks. The complexity of building per-flow tracking into a router also hinders overall performance – a penalty innocent sources may be unwilling to pay.

Reactive router-based DoS defense mechanisms are also vulnerable to low-rate attacks [80], where the attack maintains a sending rate below the threshold for attack detection, thereby avoiding the countermeasures altogether. Pushback is vulnerable to attacks that maintain the drop rate of a target router just below the attack classification threshold. Flow-based schemes (i.e. REDPD) can be subverted with bursty on-off attack schemes, where the on period is short and the off period is long. These attacks mainly focus on generating TCP congestion, which forces non-malicious TCP senders to back off. These attacks however use probing traffic to get

feedback from the network to determine, in the Pushback case, the drop rate of a particular link. This probing traffic is by definition heavily asymmetric and will be severely punished by a symmetry limiter.

5.4 Proactive defenses

TCP service protection is explored in [29] and [20]. Both techniques push the TCP handshake away from the target to perimeter edge-routers of the victims ISP, where greater bandwidth and redundancy can potentially absorb an attack. In [29], a database stores a hash of the path (router hops) taken by the TCP SYN packet to allow remaining traffic along that path to pass through the perimeter routers unabated. However, attackers can piggyback flooding traffic on a validated path already stored in the database. In CAT [20], the victim's direct ISP is not necessarily assumed to be provisioned enough to absorb an attack. Peered ISPs cooperate to push the TCP handshake to an ISP along the path that has the necessary provisioning. CAT hinges on the assumption that ISPs in an economic relationship are mutually trusting. While there exists indirect economic incentive for transitively connected ISPs, political momentum and startup investment, as well as the modifications required to BGP, inhibit the acceptance and deployability of the solution. Lastly, both these solutions only provide protection for connection exhaustion of TCP services – bandwidth floods remain a real and viable threat.

The use of routing overlay networks to combat DoS attacks is explored in [45, 4, 49]. In SOS [45], a target only communicates directly with a small number of secret servlet nodes. Routers near the target are configured to accept traffic only from said secret servlets' IP addresses. A hash func-

tion then determines the machines in the overlay that will act as the secret beacon nodes – the only other nodes that know the identities of the secret servlet nodes. Clients are authenticated (SSL or TLS) at designated entry points into the routing overlay, and the overlay eventually delivers the request through a beacon, then a servlet, and finally to the target. Mayday [4] generalizes SOS with a filter ring of routers located at the core-edge boundary of the target's network, which are routed to by the overlay nodes. These systems count on strong assumptions regarding the secrecy of all servlets and beacons in the network. Protecting the target machine by keeping the target's address information secret advocates security-through-obscurity – a well known blunder in designing secure systems. Once an attacker compromises the servlet or target's IP address, the malicious host(s) can simply send packets directly, skipping the overlay altogether. In [49], end-hosts specify the characteristics of traffic they are willing to accept, relying on well-provisioned edge routers to perform filtering on their behalf. An end-host is represented by a public and private identifier; the public identifier is widely distributed for general use, and the private identifier is known only to the routing overlay. Like SOS and Mayday, compromise of the private identifier or simply the IP address of the target renders the system completely useless, as attackers can send arbitrary traffic directly to the victim.

Firebreak [32] provides DDoS protection by taking IP reachability out of the hands of end-hosts, forcing end hosts to tunnel communication through *firebreak boxes*. The approach improves upon previous routing overlays, as attackers cannot simply skip the overlay to send IP packets directly to the protected host – such packets are consumed by routing/forwarding black-holes within the firebreak fabric. However, the approach remains a security-by-obscurity solution, as an attacker now simply requires a non-firebreak

monitored connection, and the knowledge of at least one of the firebreak machine's unicast addresses. With this knowledge, the attacker can forge packets as if they originated from a firebreak node, and successfully flood the link of protected hosts. The authors also admit the solution requires significant initial deployment to be effective.

PRIMED applies the community-of-interest approach to DoS protection [74], providing a framework to allow customers to specify good and bad communities of interest. Here, a community of interest is a portion of the IP address space and the end host categorizes communities on their innocence or malice. The end-host also specifies traffic regulation policy for each community based on the community categorizations. However, the "good clients" are identified using heuristics during an initial learning or quarantine phase. It remains unclear as to whether such a learning phase could be completely free of malicious traffic, or whether a finite phase could ever identify *all* innocent clients. Admittedly, the solution perpetuates an arms race in the development of heuristics to discern good from bad clients and the counter-attacks for attackers to fool said heuristics.

5.5 Sophisticated DoS attacks

5.5.1 Cunning Attacks

Reflector attacks [60] occur when the attacking machines coerce high-capacity servers into blasting the victim on their behalf. Attackers send forged TCP data requests to the server using the victim's IP address as the source address, causing the destination server to send the reply traffic to the victim – not back to the attacker. Well-crafted attack packets requesting large objects from the servers can elicit massive amounts of traffic for the unfor-

tunate victim. Such attacks are attractive (from the attacker's perspective) since there are a high-number of potential reflector servers throughout the internet with high bandwidth links. Fewer slaves are required when one slave can elicit flooding from several well-provisioned servers. The reflector also provides a layer of indirection between the victim and the source of attack (the slaves), thus increasing the difficulty in tracing back to the attack source. However, since all reply traffic is reflected to the target victim, the forged request traffic will be subject to symmetry limiting, thereby reducing the attack strength. As well, attackers must be able to spoof source address information to trick the reflector into sending reply traffic to the victim – which is also preventable with a symmetry limiter.

Low-rate denial of service attacks are examined in [48] and [80]. The key insight is that well-timed short-lived but high-rate bursts can trick TCP senders into falsely detecting congestion and forcing senders into perpetual congestion recovery. The attacks claim to be low-rate since the average attack rate is low due to the relatively lengthy idle periods between each high-rate burst. However, these short-but-high-rate bursts are are heavily asymmetric and would be severely limited by a symmetry limiter, crippling the potency of such attacks.

5.5.2 Protocol Vulnerabilities

A TCP receiver can take advantage of the TCP vulnerabilities discussed in [64] to cause a TCP sender to inject unreasonably large bursts of data into the network, resulting in severe congestion collapse [68]. ACK-splitting causes a sender to rapidly expand the congestion window far beyond the bottleneck capacity of the network. Similarly, optimistic acknowledgements

cause a sender to both expand the congestion window and transmit massive chunks of new data using the enlarged congestion window, even when previous data has not been received at the destination end. These attack vectors are unfortunate vulnerabilities of TCP itself, and end-hosts must tune their TCP sending behaviour to defend against such attacks. A symmetry limiter at the malicious receiver's network does provide additional benefit for the sender, keeping the optimistic acknowledgement increase to a linear factor, rather than exponential. Moreover, when a sender detects this malicious receiver behaviour, the symmetry limiter enables the sender to throttle the malicious receiver simply by discontinuing communication with that host.

Source routing in IPv4 allows a sender to create a routing loop in the network by repeating the same addresses in the source route. The attacks are known as Routing Header Type 0 or RH0 attacks, and the best solution remains to simply reject any source-routed packets. As such, there is significant momentum for the feature to be disable or removed from IPv6 [1]. As these routing loops can be caused by a small number of packets, symmetry limiting does not aid in combating such attacks.

5.6 Drastic Measures

As existing protocols and network infrastructure were not designed with security in mind and due to the prevalence of malicious activity on the Internet, it has become somewhat fashionable in the research community to consider modifying standardized protocols and even building completely new Internet architectures from the ground up, with security as a principle design goal. However, much like the previous solutions discussed, the critical mass of political and economic backing required to get such solutions widely

deployed significantly hinders their viability.

5.6.1 New Network architectures

In [37], the authors overhaul network addressing to mitigate the threat of denial-of-service attacks. Key changes to addresses include hop-by-hop domain name addressing, separate address spaces for clients and servers, forbidding client-to-client and server-to-server communication, and adding a state-setup bit to the IP protocol. The hop-by-hop addressing prevents source address spoofing and simplifies Pushback-esque techniques. The state setup bit is meant to aid filtering of connection exhaustion attacks (i.e. TCP SYN floods). Forbidding client-to-client or server-to-server communication aims to slow virus and worm propagation, forcing a layer of indirection out of the monoculture. However, forbidding communication between two types of machines drastically reduces flexibility of network evolution, and moreover, viruses and worms can still spread via tunneled communication – a worm needs only one compromised client and one compromised server in order to compromise any remaining machine. Although addresses would be traceable, a machine is still able to send a flooding attack to any other host on the network. And the accountability argument does not stand, since current DDoS attacks do not spoof source addresses [53], as the true attacker's identity is hidden by the botnet already.

Capability-based schemes [81, 6] prevent an end host from sending unwanted traffic to a destination using cryptographically secure capabilities, where a capability is granted by the destination to a particular sender. Capability-carrying packets can be verified at each capability-enabled router along the path, providing defense-in-depth against abusive sources that may

be attempting to consume more bandwidth than they have been explicitly granted. A small portion of bandwidth is allocated for end-hosts to negotiate capabilities, and as part of an incremental deployment strategy, this bandwidth is shared by legacy (non-capability-aware) traffic. Unfortunately, the shared capability-acquisition and legacy-traffic channel is left vulnerable to attack. Flooding this channel results in a denial-of-capability (DoC) attack, preventing new connections from being established as new capabilities cannot be negotiated.

Currency based schemes [75, 9] attempt to bootstrap the DoC problem, forcing senders to "pay" for the consumption of destination resources via the consumption of their own bandwidth or computing power. DoS-resistant key exchange protocols [2] establish capabilities in a similar DoC resistant manner. However, such defense remains governed by the currency possessed by the requestor (or the attacker), not the provisioning of the victim host. Attackers can consume a greater portion of the victim's resources by obtaining a greater share of the currency, either bandwidth or computing power. Furthermore, these techniques are geared towards resource exhaustion attacks and provide no defense against pure bandwidth flooding attacks. On the contrary, a symmetry limiter is a practical mechanism to protect a capability negotiation channel from DoS floods, thereby preventing DoC attacks. Widely deployed symmetry limiters could protect the capability channels using a tight threshold specific to the capability negotiation protocol.

Similar to packet symmetry, predicate routing [63] defines what is allowed or *good* in the network and rejects everything else. The network topology is modelled in terms of uni-directional links of source-destination pairs. Associated with each link are boolean disjunctive predicates that specify which attributes of packets are allowed (or disallowed) to traverse

the link. Attributes include protocol fields such as IP addresses, protocol type, and port numbers. The predicates enable any node to locally deduce a packet's path simply from the packet's attributes. Local filtering decisions can be made based on this deduced and genuine path information. However, to make a local decision, the node must obtain all of the predicates in the network and run a linked-state-like algorithm to determine packet paths. As well, once a packet reaches a non-predicate-enabled router, all bets are off as to the integrity of any inferences on packet origin and path.

SANE [21] bolts a secure routing framework on to the data link layer, as a solution to secure services for enterprise networks. A centralized Domain Controller (DC) hands out capabilities (encrypted address routes) for services on the network. Switches and end hosts never learn the network topology – each node is privy only to its neighbours addresses due to the encrypted route data. A host must authenticate with the DC before it can acquire a capability. Once authenticated, the DC knows the machine's exact location and can mitigate flooding attacks via rate limiting or capability revocation. However, this solution targets enterprise networks and requires significant hardware upgrade, a cost that may deter enterprises from this approach.

5.6.2 Protocol Modifications

The brittleness of the Internet routing infrastructure is surveyed in [55]. As end-host software continues to be patched, routing attacks are suspected to become more common as a means to disrupt the Internet. Of greatest importance, security enhancements to routing protocols often sacrifice performance for better security, leading to impractical solutions for the real

world. Future improvements to the routing infrastructure must ensure the cryptography used is fast to make deployment feasible. De-coupling of control and data channels would also improve security, preventing data traffic from masquerading as control traffic. Also, routing security decisions are strictly boolean (i.e. allow or deny), where as a continuous scale of trust would make the protocols more flexible and robust. Firewalls and intrusion detection systems should also be integrated into the routing system, providing greater defense-in-depth. But again, all such improvements must focus on high performance requirements in order to make deployment feasible. As previously discussed, the symmetry limiting mechanism is fundamentally very simple – simple enough to be implemented in hardware or FPGA. From this, and the performance results of the Linux prototype, a symmetry limiter built in specialized hardware seems a viable approach to achieving an extremely high-performance solution.

Re-feedback [17] and explicit congestion notification (re-ECN) [18] explore simple modifications to IPv4 and IPv6 protocols that enable a network link to make accurate and truthful predictions regarding potential congestion of the remaining upstream path. These modifications enable ECN policers anywhere along the path to enforce QoS or DDoS protection policies. This requires a sender (or other upstream node) to set the ECN-bit(s) in the IP header when a congestion event occurs, namely when there is lack of reply traffic. This approach simply makes congestion events explicit in the IP header – signaling which is implicitly inferred by a symmetry limiter without requiring any protocol changes.

Chapter 6

Discussion and Conclusion

6.1 Discussion

6.1.1 A Current Defense

The packet symmetry metric, as presented in this thesis, provides practical defense against malicious DoS activity and leaves the vast majority of innocent network traffic unscathed. While a handful of protocols or communication patterns break with symmetry limiting, the overall strategy is sound; link-local asymmetry, such as multicast DNS or syslog streams, can occur entirely behind a symmetry limiter; the threat model protects, rather than limits, media streaming servers sending high-rate one-way flows to clients; and limiting asymmetric ICMP messages generated by port scanning behaviour is a good security practice, slowing the scanning rates of malicious hosts. However, packet symmetry retrofits a fundamental principle on a network with protocols and applications that were not originally designed with this principle in mind. As such, the threshold values chosen in this thesis reflect the need to provide leeway to be more tolerant of innocent but less symmetric traffic patterns. In turn, these looser symmetry thresholds diminish the strength of protection – a common problem when security is implemented as an after-thought.

Notice that if security properties (such as packet symmetry) were built

into the network infrastructure, the network would be automatically self-critical. All end-hosts and routers would be forced to adhere to the security guarantees designed into the protocols they had to use to communicate with any other machine. However, building in security as after-thought leaves network operators with the *option* to implement self-critical filtering manually, with no requirement to implement such filtering. As such, the argument to deploy self-critical network filtering is socialist in nature, relying on the altruism of the ISP to take on the responsibility (and cost!) of filtering its own outgoing malicious traffic for the greater good. The density of broadband users has been directly correlated to the proportion of botnet enlisted machines [28], which suggests broadband ISPs are fueling botnet growth and have a social duty to protect the value of the network by combating this malicious activity. Unfortunately, altruism is not an abundant quality in many corporations with their main focus on maximizing profits. This leads back to the question of what direct financial incentive could motivate an ISP to deploy a symmetry limiter?

The momentum to deploy self-critical networking filters suffers from a catch-22 – many deployments will lead to further deployments, but no deployments prevents anyone from deploying. Ingress filtering once suffered from the same vicious cycle [36], though eventually the critical mass of momentum was formed as ISPs realized enforcing source address integrity *actually helped* in administering their own network [72], and today over 80% of net block addresses are non-spoofable (i.e. ingress filtered) [15]. Unfortunately, symmetry limiting lacks this direct incentive by adding complexity to network administration, since innocent (i.e. non-DoS) packets can be dropped and cause some applications (ex. peer-to-peer file sharing) to malfunction. Though the ISP might like to limit such asymmetric traffic in

the first place, the provider must cater to the demands of the consumer and provide connectivity that allows such applications to function correctly, otherwise the consumer will simply switch to a different and more permissive ISP.

One concrete financial incentive for an ISP to deploy self-critical packet symmetry limiters could come from the legal community, if the calls to update computer crime legislation to specifically outlaw DoS attacks are answered [66, 44, 52]. Presently, computer crime legislation does not specify DoS attacks as a crime, preventing prosecutors from securing convictions even when the perpetrators can be identified [41]. If DoS attacks were made illegal, not only could the attackers be convicted, but the attack victims would have the legal grounds on which to sue for damages incurred as a result of the attack. Sorting out who is liable for the damages of a botnet-based DoS attack may be less clear – is the botnet mastermind solely responsible for the attack, or are the zombie machine owners partly responsible for a lack of following network security best-practices and allowing their machine to be compromised? In this light, self-critical symmetry limiting could provide both the subscribers and the ISP with the clear financial benefit of being absolved of legal liability for DoS attacks perpetrated by their subscribers, since the contribution of each subscriber to a flooding attack is at most the packet bootstrapping rate P_B (see Section 2.1.3). Following similar logic, spammers have been arrested and taken to court [61, 69], with perpetrators typically charged under extortion or fraud legislation. Nevertheless, perhaps in an effort to avoid liability concerns, AOL, one of the largest North American service providers, now takes self-critical action on its outgoing email to reduce the amount of spam generated by its subscribers [7]. Note that for symmetry limiting to absolve an organization of negligence, the method-

ology (including the parameter values to establish P_B) would have to be recognized as a best-practice by the networking community. This recognition would require a critical mass of political momentum to bolster the packet symmetry solution, which may be further complicated with support for the many other DoS protection and prevention solutions proposed in the past (see Chapter 5).

6.1.2 Future Network Architectures

Initiatives to develop the next-generation Internet architecture [34] are generating significant activity in the research community. The related work chapter of this thesis notes several proposals for new DoS-resistant architectures, using new cryptographically secure protocols, building explicit signaling and control directly into the network fabric. However, these initiatives are clearly not focussed solely on DoS attack prevention. A complete redesign of the network presents the opportunity to consider several key and fundamental design properties of the network, including how to specify names, how to organize routing to support mobility, multi-homing and network heterogeneity, as well as the direct integration of authentication, access control and other security paradigms. As the Internet may well be in store for a complete and total redesign, that begs the question; does packet symmetry have a role in the network architecture of the future?

The analysis in this thesis suggests that symmetry should be a fundamental design property of network protocols and as such be an integral part of a next-generation Internet. However, not all protocols need to implement perfectly symmetric communication – mainly signaling and control protocols should be designed to exhibit symmetry to a very high degree (i.e. 1:1) to al-

low strong protection for session-establishing control channels. This prevents denial-of-capability attacks that flood a control channel with bogus requests, blocking innocent clients from establishing new connections to protected services. Note as well that the metric need not be packet symmetry specifically, as the next generation network may not even be packet switched. Fundamentally, a generic symmetry limiting mechanism that enforces a balance of outgoing-to-incoming *traffic units* should be widely deployed at the network edge. These modules cannot be deployed on the end host since the end host cannot be trusted – even with virtual machine systems, trusted computing platforms, and tamperproof hardware, a client always has ultimate control over their machine, and could thus subvert a protection mechanism. Therefore, while the metric may not be exactly as described here, nor the thresholds the same, symmetry seems a fundamentally useful property to consider for modern network design.

6.2 Conclusion

This thesis evaluated and further developed packet symmetry [47] as a proactive source-based filtering mechanism to prevent DoS attacks. The packet symmetry metric captures the implicit signaling already present in network communication protocols and applications, requiring no changes to end hosts or routing infrastructure for deployment. Analysis of real network traces lead to the development of a packet symmetry threshold that is effective in discerning good from malicious traffic with extremely few false-positives. A thorough security evaluation showed that symmetry-limiting defends well against flooding DoS attacks and enables victims to manage defense against resource-exhaustion attacks with local administration. Fur-

ther analysis demonstrated the mechanism's own resilience against attacks.

The symmetry limiter prototype for the Linux kernel proved the feasibility of the approach, both in terms of performance and usability. Deployment on a network bridge allows a symmetry limiter to be easily incorporated into a network without requiring any modifications to end hosts or routers. The solution is immediately deployable and provides incremental benefit with increased deployment. Widely deployed symmetry limiting may serve as both a temporary solution for the current Internet architecture, as well as a corner stone for a next-generation secure network architecture, bootstrapping protection for control or capability-acquisition channels.

Bibliography

[1] J. Abley Afilias and P. Savola. Deprecation of type 0 routing headers in IPv6. May 2007. Internet draft, work in progress.

[2] William Aiello, Steven M. Bellovin, Matt Blaze, John Ioannidis, Omer Reingold, Ran Canetti, and Angelos D. Keromytis. Efficient, DoS-resistant, secure key exchange for internet protocols. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 48–58, New York, NY, USA, 2002. ACM Press.

[3] M. Allman, V. Paxson, and W. Stevens. RFC 2581: TCP congestion control, April 1999.

[4] D.G. Andersen. Mayday: Distributed Filtering for Internet Services. *4th USENIX Symposium on Internet Technologies and Systems USITS*, 2003.

[5] Starr Andersen. Changes to functionality in Microsoft Windows XP Service Pack 2: Part 2: Network protection technologies. http://technet.microsoft.com/en-us/library/bb457156.aspx#EHAA, cited Jul 22, 2007.

[6] Tom Anderson, Timothy Roscoe, and David Wetherall. Preventing internet denial-of-service with capabilities. *SIGCOMM Comput. Commun. Rev.*, 34(1):39–44, 2004.

[7] AOL port 25 faq. http://postmaster.aol.com/faq/port25faq.html, cited Aug 1, 2007.

[8] K. Argyraki and D.R. Cheriton. Active internet traffic filtering: real-time response to denial-of-service attacks. *Proceedings of the General Track. 2005 USENIX Annual Technical Conference*, pages 135 – 48, 2005.

[9] T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. *Proceedings of the 8th International Workshop on Security Protocols, Lecture Notes in Computer Science, Cambridge, UK, April*, 2000.

[10] Paul Bacher, Thorsten Holz, Markus Kotter, and Georg Wicherski. Know your enemy: Tracking botnets. http://www.honeynet.org/papers/bots, March 2005.

[11] BBC News: Blackmailers target $1m website. http://news.bbc.co.uk/1/hi/technology/4621158.stm, cited Aug 1, 2007.

[12] S. Belovin, M. Leech, and T. Taylor. ICMP traceback messages, March 2000. Internet draft, work in progress.

[13] Scott Berinato. Attack of the bots. http://www.wired.com/wired/archive/14.11/botnet.html?pg=1&topic=botnet&topic_set=, cited Aug 1, 2007.

[14] Scott Berinato. How a bookmaker and a whiz kid took on an extortionist - and won. http://www.csoonline.com/read/050105/extortion.html, cited Aug 1, 2007.

[15] Robert Beverly. ANA Spoofer Project - State of IP Spoofing. http://spoofer.csail.mit.edu/summary.php, cited Aug 14, 2007.

[16] Robert Beverly and Steven Bauer. The Spoofer Project: Inferring the Extent of Source Address Filtering on the Internet. *USENIX SRUTI: Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages pp. 53–59, July 2005.

[17] Bob Briscoe, Arnaud Jacquet, Carla Di Cairano-Gilfedder, Alessandro Salvatori, Andrea Soppera, and Martin Koyabe. Policing congestion response in an internetwork using re-feedback. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 277–288, New York, NY, USA, 2005. ACM Press.

[18] Bob Briscoe, Arnaud Jacquet, Alessandro Salvatori, and Martin Koyabe. Re-ECN: Adding accountability for causing congestion to TCP/IP. *IETF Internet-Draft*, October 2006.

[19] R. Bush, D. Karrenberg, M. Kosters, and R. Plzak. RFC 2870: Root name server operational requirements, June 2000.

[20] M. Casado, A. Akella, Pei Cao, N. Provos, and S. Shenker. Cookies along trust-boundaries (CAT): accurate and deployable flood protection. pages 15–22, San Jose, CA, USA, 2006.

[21] Martin Casado, Tal Garfinkel, Aditya Akella, Michael Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. SANE: A protection architecture for enterprise networks. *Proceedings of the 15th USENIX Security Symposium*, August 2006.

[22] I. Ceaparu, J. Lazar, K. Bessiere, J. Robinson, and B. Shneiderman. Determining causes and severity of end-user frustration. *International Journal of Human-Computer Interaction*, 17(3):333 – 56, 2004//.

[23] S. Cheshire and M. Krochmal. Multicast DNS. *IETF*, August 2006. Internet draft - work in progress.

[24] Stuart Cheshire. Zero configuration networking (zeroconf). http://www.zeroconf.org/, cited Jun 19, 2007.

[25] K. Cho, K. Mitsuya, and A. Kato. Traffic Data Repository at the WIDE Project. *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, 2000.

[26] CompleteWhoIs - Using IP Lists. http://www.completewhois.com/ bogons/bogons_usage.htm, cited Jul 2, 2007.

[27] Tom Espiner. CPS pushing for teen DoS trial to return to court. http://management.silicon.com/government/0,39024677,39155404,00.htm?r=2, cited Aug 2, 2007.

[28] Dean Turner et al. Symantec Internet Security Threat Report. Technical Report XI, Symantec Corporation, March 2007. http://eval.symantec.com/mktginfo/enterprise/white_papers/ent-whitepaper_internet_security_threat_report_xi_03_2007.en-us.pdf.

[29] Hikmat Farhat. Protecting TCP services from denial of service attacks. In *LSAD '06: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 155–160, New York, NY, USA, 2006. ACM Press.

[30] Rik Farrow. DNS root servers: Protecting the internet. http://www.spirit.com/Network/net1102.html, cited Aug 1, 2007.

[31] P. Ferguson and D. Senie. RFC 2827: Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing, May 2000.

[32] P. Francis. Firebreak: An IP Perimeter Defense Architecture. *Cornell University Library*, December 2006. http://hdl.handle.net/1813/5753.

[33] Lee Garber. Denial-of-service attacks rip the internet. *Computer*, 33(4):12–17, 2000.

[34] GENI: Global environment for network innovation. http://www.geni.net/news.html, cited Jul 5, 2007.

[35] Thomer M. Gil and Massimiliano Poletto. MULTOPS: A data structure for bandwidth attack detection. *10th USENIX Security Symposium*, August 2001.

[36] A. Greenhalgh, M. Handley, and F. Huici. Using Routing and Tunneling to Combat DoS Attacks. *Proc. Usenix workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2005.

[37] Mark Handley and Adam Greenhalgh. Steps towards a DoS-resistant internet architecture. In *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 49–56, New York, NY, USA, 2004. ACM Press.

[38] C. Hornig. RFC 894: Standard for the transmission of IP datagrams over Ethernet networks, April 1984.

[39] ICANN factsheet - root server attack on 6 February 2007. http://www.icann.org/announcements/factsheet-dns-attack-08mar07.pdf, cited Aug 1, 2007.

[40] J. Ioannidis and S.M. Bellovin. Implementing pushback: router-based defense against DDoS attacks. *Ninth Annual Symposium on Network and Distributed System Security*, February 2002.

[41] Is Canada losing the fight against online thieves? http://www.cbc.ca/news/background/tech/online-crime-war.html, cited Aug 2, 2007, May 2007.

[42] J.A. Jacko, A. Sears, and M.S. Borella. The effect of network delay and media on user perceptions of web resources. *Behaviour and Information Technology*, 19(6):427 – 39, November 2000.

[43] Cheng Jin, Haining Wang, and Kang G. Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 30–41, New York, NY, USA, 2003. ACM Press.

[44] M. E. Kabay. Distributed denial-of-service attacks, contributory negligence and downstream liability. *Ubiquity*, 1(2):3, 2000.

[45] A.D. Keromytis, V. Misra, and D. Rubenstein. SOS: secure overlay services. *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 61–72, 2002.

[46] S.T. King and P.M. Chen. Subvirt: implementing malware with virtual machines. *IEEE Symposium on Security and Privacy*, page 14 pp., May 2006.

[47] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt. Using packet symmetry to curtail malicious traffic. *HotNets: Proceedings from the Fourth Workshop on Hot Topics in Networks*, 12, 2005.

[48] Aleksandar Kuzmanovic and Edward W. Knightly. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86, New York, NY, USA, 2003. ACM Press.

[49] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, and Ion Stoica. Taming IP packet flooding attacks. *SIGCOMM Comput. Commun. Rev.*, 34(1):45–50, 2004.

[50] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.*, 32(3):62–73, 2002.

[51] MANAnet reverse firewall. http://www.cs3-inc.com/ps_rfw.html, cited Aug 1, 2007.

[52] Many countries said to lack computer crime laws. http://archives.cnn.com/2000/TECH/computing/07/26/crime.internet.reut/index.html, cited Aug 2, 2007.

[53] Z. Morley Mao, Vyas Sekar, Oliver Spatscheck, Jacobus van der Merwe, and Rangarajan Vasudevan. Analyzing large DDoS attacks using mul-

tiple data sources. In *LSAD '06: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 161–168, New York, NY, USA, 2006. ACM Press.

[54] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the source. *Proceedings 10th IEEE International Conference on Network Protocols*, pages 312 – 21, 2002.

[55] D. Montgomery and S. Murphy. Toward secure routing infrastructures. *IEEE Security and Privacy*, 4(5):84 – 7, Sept.-Oct. 2006.

[56] David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, 2006.

[57] Optimize vista for BitTorrent, eMule, P2PTV. http://torrentfreak.com/optimize-vista-for-bittorrent-emule-p2ptv/, cited Jul 22, 2007.

[58] Vern Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Trans. Netw.*, 5(5):601–615, 1997.

[59] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[60] Vern Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *SIGCOMM Comput. Commun. Rev.*, 31(3):38–47, 2001.

[61] Pharmacy spam king rizler sentenced to 30 years. http://www.govtech.com/gt/128786?topic=117677, cited Aug 10, 2007.

[62] Rogers high-speed internet services. http://www.rogers.ca, cited Jul 6, 2007.

[63] Timothy Roscoe, Steve Hand, Rebecca Isaacs, Richard Mortier, and Paul Jardetzky. Predicate routing: enabling controlled networking. *SIGCOMM Comput. Commun. Rev.*, 33(1):65–70, 2003.

[64] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. *SIGCOMM Comput. Commun. Rev.*, 29(5):71–78, 1999.

[65] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for IP traceback. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 295–306, New York, NY, USA, 2000. ACM Press.

[66] Michael B. Scher. On doing 'being reasonable'. *;login*, December 2006.

[67] L.M. Schleifer and B.C. Amick III. System response time and method of pay: stress effects in computer-based tasks. *International Journal of Human-Computer Interaction*, 1(1):23 – 39, 1989.

[68] Rob Sherwood, Bobby Bhattacharjee, and Ryan Braud. Misbehaving TCP receivers can cause internet-wide congestion collapse. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 383–392, New York, NY, USA, 2005. ACM Press.

[69] Six held over net scam. http://www.australianit.news.com.au/story/0,24897,22214435-15306,00.html, cited Aug 10, 2007.

[70] Skype. http://www.skype.com, cited Apr 15, 2007.

[71] SSAC Advisory SAC008 DNS Distributed Denial of Service (DDoS) Attacks. http://www.icann.org/committees/security/dns-ddos-advisory-31mar06.pdf, cited Aug 1, 2007.

[72] Telus inc., April 2007. Personal correspondance.

[73] Verizon high-speed internet. http://www.verizon.com, cited Jul 6, 2007.

[74] Patrick Verkaik, Oliver Spatscheck, Jacobus Van der Merwe, and Alex C. Snoeren. Primed: community-of-interest-based ddos mitigation. In *LSAD '06: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 147–154, New York, NY, USA, 2006. ACM Press.

[75] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. Ddos defense by offense. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 303–314, New York, NY, USA, 2006. ACM Press.

[76] Jianping Wang. Traffic regulation under the percentile-based pricing policy. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 4, New York, NY, USA, 2006. ACM Press.

[77] Member-Haining Wang, Member-Danlu Zhang, and Fellow-Kang G. Shin. Change-point monitoring for the detection of DoS attacks. *IEEE Trans. Dependable Secur. Comput.*, 1(4):193–208, 2004.

[78] XiaoFeng Wang and Michael K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 257–267, New York, NY, USA, 2004. ACM Press.

[79] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[80] Y. Xu, R. Guerin, and R. Guerin. On the robustness of router-based denial-of-service (DoS) defense systems. *Computer Communication Review*, 35(3):47 – 60, July 2005.

[81] Xiaowei Yang, David Wetherall, and Thomas Anderson. A DoS-limiting network architecture. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 241–252, New York, NY, USA, 2005. ACM Press.

[82] D.K.Y. Yau, J.C.S. Lui, and Feng Liang. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. *IEEE 2002 Tenth IEEE International Workshop on Quality of Service (Cat. No.02EX564)*, pages 35 – 44, 2002//.