Statistical Approaches to DDoS Attack Detection and Response[1]

Laura Feinstein, Dan Schnackenberg
The Boeing Company, Phantom Works
Laura.C.Feinstein@boeing.com
Daniel.D.Schnackenberg@boeing.com

Ravindra Balupari, Darrell Kindred
Network Associates Laboratories
Ravindra_Balupari@nai.com
Darrell_Kindred@nai.com

Abstract

The nature of the threats posed by Distributed Denial of Service (DDoS) attacks on large networks, such as the Internet, demands effective detection and response methods. These methods must be deployed not only at the edge but also at the core of the network. This paper presents methods to identify DDoS attacks by computing entropy and frequency-sorted distributions of selected packet attributes. The DDoS attacks show anomalies in the characteristics of the selected packet attributes. The detection accuracy and performance are analyzed using live traffic traces from a variety of network environments ranging from points in the core of the Internet to those inside an edge network. The results indicate that these methods can be effective against current attacks and suggest directions for improving detection of more stealthy attacks. We also describe our detection-response prototype and how the detectors can be extended to make effective response decisions.

1. Introduction

Powerful DDoS toolkits are available to potential attackers, and essential networks are ill prepared for defense. The security community has long known that DDoS attacks are possible, but only in the past three years have such attacks become popular with hackers. As ominous as the threat is today, it will only worsen as tools are built to evade defenses. Soon, DDoS floods will appear that are difficult to distinguish from legitimate traffic, and packet rates from individual flood sources will be low enough to escape notice by local administrators. To meet the increasing need for detection and response, researchers face these major issues:

- A stand-alone router on the attack path should automatically recognize that the network is under attack and adjust its traffic flow to ease the attack impact downstream.
- The detection and response techniques should be adaptable to a wide range of network environments, preferably without significant manual tuning.
- Attack detection should be as accurate as possible. False positives can lead to inappropriate responses that cause denial of service to legitimate users. False negatives result in attacks going unnoticed.
- Attack response should employ intelligent packet discard mechanisms to reduce the downstream impact of the flood while preserving and routing the non-attack packets.
- The detection method should be effective against a variety of attack tools available today and also robust against future attempts by attackers to evade detection.

These are demanding goals, but we contend that there are several reasons to believe that satisfactory detection and response methods can be designed. DDoS traffic generated by today's tools often has packet-crafting characteristics that make it possible to distinguish from normal traffic. For example, in some configurations the Stacheldraht attack tool crafts packets so that the source port is random and the destination port is sequentially increased from one packet to the next [1],[10]. Future DDoS tools may include improvements to packet crafting. However, we claim that these tools are unlikely to model legitimate traffic closely enough to produce crafted packets that do not distort statistical measurements of the composition of the traffic. Our hypothesis is that relatively simple statistical measures can be used to discriminate DDoS traffic from legitimate traffic in core routers with sufficient accuracy to mitigate the effect of the attack downstream.

Research conducted by other organizations suggests that statistical measurements and statistical processing are an effective approach to the DDoS problem. The EMERALD project at SRI International uses intrusion

[1] This research was supported by DARPA under contract N66001-01-C-8048.

detection signatures with Bayesian inference to detect distributed attacks [12].

Researchers at Florida Institute of Technology have created an intrusion detection system (IDS) that is non-stationary and models probabilities based on time since the last event rather than on average rate [6]. This IDS operates on many of the same fields our detector monitors and has similar training requirements to set up initial thresholds and baselines. The system has two components, PHAD and ALAD. PHAD operates on the packet header while ALAD operates on an incoming server TCP connection. The PHAD component clusters observed values and then compares the size of the clusters to accepted thresholds to determine anomalies.

Mazu Networks uses a similar architecture to PHAD and our chi-square detector. The Mazu system collects network statistics through a monitoring device and similarly sorts the collected items into buckets [3]. An algorithm determines whether buckets should be divided or combined and a threshold detects anomalies depending on the number and size of the buckets.

We have imposed some significant constraints on our DDoS defense development: no explicit coordination (e.g., *pushback* [7]) between defending network components, no built-in knowledge of applications or protocols, and no instrumentation at end hosts. These approaches are being actively explored in other research, and we believe that the techniques described here can complement these others in a comprehensive DDoS defense solution.

2. Detection Algorithms

Our detection algorithms measure statistical properties of specific fields in the packet headers at various points in the Internet. For instance, if a detector captures 1000 consecutive packets at a peering point and computes the frequency of occurrence of each unique source IP address in those 1000 packets, then the detector will have a model of the distribution of the source address. Further computations with this distribution allow us to measure the randomness or uniformity of the addresses as well as the "goodness-of-fit" of the distribution with respect to prior measurements.

2.1. Entropy

Let an information source have n independent symbols each with probability of choice p_i. Then, the entropy H is defined as [17]:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Hence, entropy can be computed on a sample of consecutive packets. Comparing the value for entropy of some sample of packet header fields to that of another sample of packet header fields from the same peering point provides a mechanism for detecting changes in the randomness. We have observed through experimentation that while a network is not under attack, the entropy values for various header fields each fall in a narrow range. While the network is under attack with current attack tools, these entropy values exceed these ranges in a detectable manner.

The algorithm to compute entropy can be optimized to perform only a few simple computations per packet. In our implementation, the entropy of a source will be calculated through a sliding window of fixed width, W. The probability value p_i in this algorithm is actually the frequency of occurrence of each unique symbol divided by the total number of symbols in the sample. The process of computing entropy of W packets is as follows:

1. Compute the entropy of the first W packets with reference to a specific header parameter (e.g. source IP address).
2. Isolate the term in the summation corresponding to the probability of the first symbol in the window (label this symbol with i=1) and also the value for the corresponding probability (p_{i-1}).
3. Slide the window so the new first term was previously the second term and the next W-1 consecutive terms are contained in the window.
4. Isolate the term in the summation corresponding to the probability of the symbol acquired from shifting the window.
5. Subtract off the terms isolated in steps 2 and 4 from the value computed in step 1.
6. Recompute the affected probabilities for the current window of data. That is, recompute p_{i-1} and the probability of the symbol that was added by sliding the window.
7. Using the values computed in step 6, add the two terms missing from the entropy summation back in and compare this new entropy value to the previous entropy computations.
8. Repeat steps 2-7 to determine subsequent entropy values.

A sophisticated attacker would likely attempt to defeat the detection algorithm by creating *stealthy* traffic floods that mimic the legitimate traffic the detector would expect. An attacker who knew that the entropy of various packet attributes was being monitored could build an attack tool that generates floods with tunable entropy levels. Through guesswork, penetration, or trial and error, the attacker could determine typical entropy levels seen at the detector and tune the flood to match. This may not be as easy as it sounds, particularly if there are multiple detectors deployed between the flood sources and the targets, as the typical entropy values

seen by detectors in different network environments are likely to differ. Stealthy attacks are explored further in Section 3.4.

The window size, W, is a tunable parameter that controls how much smoothing of short-term fluctuations the detector will do. Increasing W will reduce the variation in entropy and may reduce the rate of false-positives resulting from brief and presumably insignificant anomalies. However, W should be kept small enough that attacks are detected quickly. We have found that a window size of 10,000 packets is a reasonable compromise in the network environments we have explored.

2.2. Chi-Square Statistic

Pearson's chi-square (χ^2) Test is used for distribution comparison in cases where the measurements involved are discrete values. For example, it could be used to test the distribution of TCP SYN flag values (0 or 1) or protocol numbers. The test works best when the number of possible values is small. In particular, a rule of thumb is that the expected number of packets in a sample having each possible value be at least five. However, this can often be achieved through "binning", that is combining a set or range of possible values and treating them as one. For example, the chi-square test can be applied to service ports by considering four values: HTTP, FTP, DNS, and "other." Similarly, packet lengths can be binned into ranges such as 0-64 bytes, 65-128 bytes, 129-255 bytes, etc.

For a sample of N packets, let B be the number of available bins. Define N_i as the number of packets whose value falls in the ith bin and n_i as the *expected* number of packets in the ith bin under the typical distribution. Then the chi-square statistic is computed as follows:

$$\chi^2 = \sum_{i=1}^{B} \frac{(N_i - n_i)^2}{n_i}.$$

When the N_i and n_i values are large and the N measurements are independent and drawn from the expected distribution, this value follows the well-known chi-square distribution with $B-1$ degrees of freedom. These assumptions (in particular, independence) do not typically hold for packet field values even under normal conditions. Hence, comparison with the chi-square *distribution* is of limited utility. However, the chi-square *statistic* does provide a useful measure of the deviation of a current traffic profile from the baseline.

A current-traffic profile, mapping packet attribute values to frequencies, is maintained as follows:

1. For each packet that arrives, extract the value, v, of the desired attribute (e.g., source address).
2. Apply exponential decay to the stored frequency for v based on its age (time since last update). The stored frequency is multiplied by
$$\exp\left(\frac{age \cdot \ln(0.5)}{halflife}\right).$$
3. Increment the frequency for v and store the current time (or packet count) as its last-update time.

Periodically, this current-traffic profile is compared with a baseline profile using the chi-square statistic, as follows:

1. Apply exponential decay to the stored current-traffic frequencies, as above.
2. Group the attribute values into bins based on frequency. For example, the 16 most common values might go in one bin, the next 64 in another, the next 256 in another, and the rest in another.
3. Calculate the total frequency for each bin.
4. Calculate the chi-square statistic, comparing these bin-frequency totals with the bin-frequency values in the baseline profile.

The baseline profile can be maintained as decaying averages of the current-traffic bin frequencies. Each time the current-traffic bin frequencies are computed, the average is updated as follows:

1. Exponential decay is applied to the stored bin-frequency averages, using a significantly longer half-life than is used for the current-traffic profile.
2. The new set of bin frequencies is multiplied by
$$1 - \exp\left(\frac{age \cdot \ln(0.5)}{baseline\_halflife}\right)$$
and the result is added to the decayed average.

The user can tune the detector by modifying the following parameters: traffic profile half-life, baseline profile half-life, bin definitions and hash function range. Values in the current-traffic profile whose frequencies decay below a certain threshold can be purged without substantially affecting the chi-square computation. This purging reduces memory consumption and processing requirements. For packet attributes such as IP addresses that have a very large range, a hash of the attribute's value may be used instead of the value itself in order to reduce memory consumption and processing requirements in the worst case (many distinct values). When the baseline frequency value for a given bin is very low, the chi-square statistic may be excessively influenced by that bin's value. Ideally, the bins will be defined such that this is unlikely, but as a fallback, low-value bins can be automatically merged with adjoining bins prior to computing the chi-square statistic.

It is unlikely that an outside attacker without access to the detector itself or a large fraction of its network neighbors will know the exact characteristics of network traffic typically seen by the detector. Therefore, we hypothesize that the attack traffic will differ from typical traffic in measurable ways.

3. Detector Evaluation

In order to evaluate thoroughly the potential effectiveness of DDoS detection methods such as those described in Section 2, we must address the following questions.

How well can the method distinguish attack conditions from normal conditions? To answer this question, we must determine what kinds of DDoS attacks the method can detect, and what fraction of the monitored traffic the attacks must comprise in order to be detected. Ideally, a detector should pick up not only attacks generated by tools found "in the wild" to date, but also more stealthy attacks using more sophisticated tools wielded by attackers familiar with the detection method and detector's network environment. Finally, we must assess the frequency and consequences of *false-positives*, ordinary fluctuations in legitimate traffic interpreted by the detector as attacks.

To what network environments and platforms is the method best suited? Characteristics of the monitored network traffic will vary significantly depending on where detectors are deployed. The protocols used, diversity of addresses seen, typical session durations, response latency, and daily volume fluctuations will differ dramatically among LAN environments, edge routers, and core routers. A detection method effective in one of these environments may fare poorly in others. In addition, if the method is to be applied in core routers, its per-packet computational requirements and memory usage must be modest in order to make real-time processing at high bandwidths practical (see Section 3.5).

Once an attack is detected, can the detector characterize the attack traffic sufficiently to produce a targeted response that mitigates the attack's effects? Detection alone may be useful for alerting human administrators to attacks in progress or notifying upstream (closer to attack sources) devices that something should be done. However, many DDoS attacks today are only two minutes in duration [8], so the ability to generate automated responses, at least as a preliminary measure, is important. A detection method that can effectively describe the nature of the attack will make such automated response more practical.

The remainder of this section describes attempts answer these questions for the entropy and chi-square DDoS detection methods.

3.1. Prototype Implementation

To evaluate the DDoS attack detection methods described in Section 2 under realistic conditions, we implemented prototype detector modules as plug-ins for Snort, the popular, open-source network intrusion detection system [13], [14]. In addition to real-time traffic monitoring, Snort supports off-line processing of previously captured network traffic, making it possible to conduct reproducible detection experiments with traffic data from a variety of environments.

The chi-square and entropy detectors were built as Snort *preprocessors*, operating on every IP datagram received by Snort prior to stream reassembly and other packet manipulation. The two detectors can be individually enabled and configured in the `snort.conf` configuration file, and can trigger alarms through Snort's modular alerting facility.

In addition to issuing alerts, these plug-ins record data to log files in the Snort log directory. The entropy detector logs periodically computed entropy values for each packet attribute specified in the initialization file (e.g., source and destination IP addresses and TCP/UDP ports, datagram length, and TCP window size). The chi-square detector logs the periodically computed chi-square statistics for each of the specified packet attributes, along with the current and baseline bin frequency values used to compute those statistics. This data can be useful for manual or automatic detector tuning and alert threshold setting.

3.2. Network Trace Data

A critical element of evaluating these detectors is exposing them to traffic from a variety of network environments. This allows us to determine how stable the traffic statistics monitored by the detectors are in those environments, and how effectively the detectors can identify DDoS attack traffic in different contexts.

For this purpose, we obtained several publicly available network traces as well as some traces collected specifically for our experiments. These traces are not known to contain substantial DDoS attacks, so we treat them as consisting of legitimate traffic. To test the effects of DDoS attacks, we simulate these attacks by overlaying the kind of attack traffic generated by some existing DDoS attack tools onto the traces at various concentrations [10]. Ideally, we would make use of traces containing identifiable periods during which actual DDoS attacks were in progress, but few of these are publicly available.

The traces used were drawn from a variety of network environments, as described below, and most have IP addresses that have been transformed via an unknown but one-to-one function for privacy purposes.

This address re-mapping is irrelevant to the currently implemented detectors, since they make no assumptions about relationships between different IP addresses.

The following traces were used:

- *NZIX*. This trace, from July 2000, includes five consecutive days of IP headers sent through the New Zealand Internet Exchange (NZIX), a peering point for several major New Zealand ISPs and the University of Waikato; throughput ranges roughly from 4 to 12 Mbits/s. Two six-hour periods were used for detector experimentation.
- *Bell Labs*. This trace contains one week of IP headers observed outside the firewall for Bell Labs, a 9Mbit/s connection serving a staff of about 450. One full day of this traffic was used for experimentation.
- *University*. This trace was collected from the Stocker Engineering and Technology network at Ohio University. It contains all the packets entering and leaving the network, with throughput ranging from 8 to 16 Mbit/s. Three sets of data, each having around 30,000,000 packets, were collected at different times during a day for experimentation.
- *Small Company*. This trace contains one week of network traffic observed outside the firewall of a small technology company in the United States. The connection served a staff of about 200 users in the company. One 24-hour weekday trace was used for experimentation.

3.3. Detection Example

To illustrate the effects of an attack on the entropy and chi-square statistics, we examined a 1,000,000-packet excerpt from the NZIX data set with a simulated DDoS attack comprising 25% of all packets, starting at packet number 700,000 and ending at packet number 800,000. (Packets in this excerpt are numbered from 200,000 to 1,200,000.) In this attack, IP source addresses are chosen at random from a uniform distribution; we will focus on source-address-based detection.

Figure 1 shows the output (entropy values) of an entropy detector examining the IP source address packet attribute with a window size of 10,000 packets. Before the attack begins, source address entropy measurements fall entirely within the range 7.0-7.5. During the attack, the entropy increases by approximately 1.5. Any maximum-entropy threshold setting between 7.5 and 8.75 would detect this attack without generating any false-positives in this example.

In Figure 2, the bin frequency profile for a source address chi-square detector (current traffic half-life is

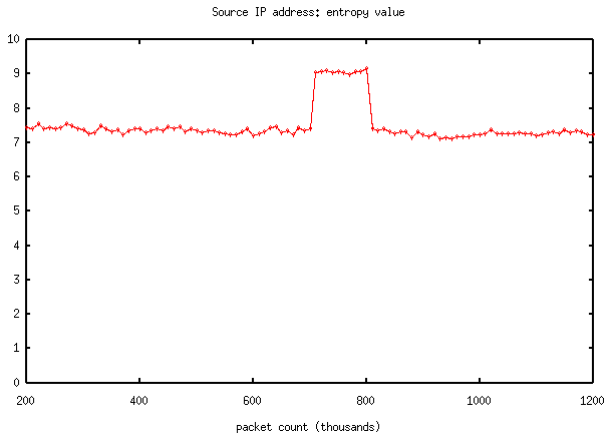

Figure 1: Entropy for a brief DDoS attack

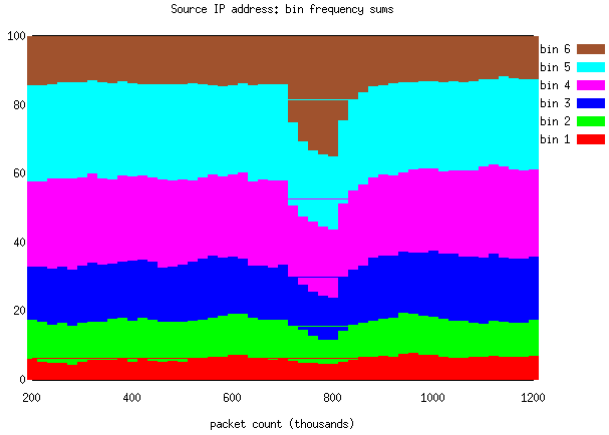

Figure 2: Bin frequencies for a brief attack

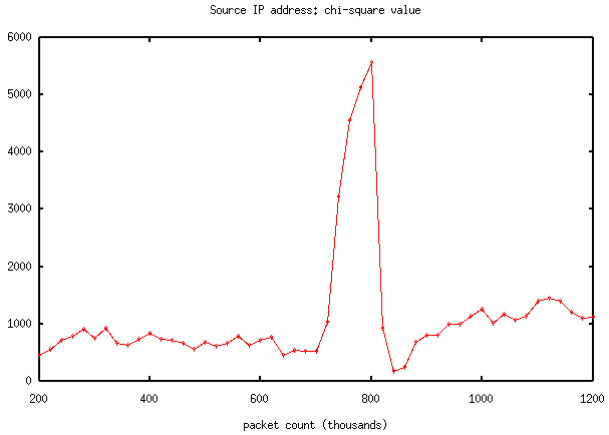

Figure 3: Chi-square values for a brief attack

20000 packets; bins defined as most frequent source address, next 4 most frequent, next 16, next 256, next 4096, and the remainder) is displayed for the same example. The six colored regions represent the percent-

age of packets falling in each of the six bins over time. Like many network characteristics [2], source address frequency for this trace follows roughly a power-law distribution, so the bins of exponentially increasing size have roughly equal frequencies. When the attack begins at packet 700,000, the total frequency of bin 6 (representing packets whose source addresses are least frequently seen) grows noticeably, as we would expect since the source addresses in the attack traffic are drawn from a uniform, rather than power-law, distribution. The chi-square values for this trace are shown in Figure 3, using a baseline profile taken from the previous day's traffic. Note the spike shortly past 700,000 packets, when the bin 6 frequency significantly exceeds the baseline value and other bin frequencies are lower than baseline. In this example, any chi-square threshold between 1500 and 5000 would catch the attack without generating false positives.

An attack in which the source addresses were fixed or drawn from a small set would produce similarly dramatic results for both entropy and chi-square detectors. The measured entropy values would drop significantly and bins 1-3 would have unusually high frequency.

3.4. Distribution of Statistics

We now look more closely at the distribution of chi-square and entropy measurements for legitimate traffic traces and for the same traces with different kinds of simulated DDoS attack traffic overlaid.

For the first set of measurements, we run source-address entropy and chi-square detectors over a six-hour period (50 million packets from 6pm-midnight local time, July 7, 2002) from the NZIX trace, using the parameter settings described in the previous section. The chi-square baseline is taken from the matching six-hour period on the previous day (July 6). We then run the same detector configuration over the same trace, with a random 25% of packets replaced with simulated DDoS attack traffic using uniformly random source addresses. The distributions of entropy and chi-square values resulting from these runs are shown in Figure 4 and Figure 5. It is clear from these histograms that the variation in entropy and chi-square statistics due to fluctuations in legitimate traffic is small when compared to the deviation caused by this DDoS attack. Therefore, with simple automatic threshold setting, both detectors will identify this attack consistently while yielding very few false-positives.

If we make more generous assumptions about the attack tools' sophistication and the attacker's knowledge, the detection task becomes significantly harder. Figure 6 show the results of repeating the previous experiment with a stealthy DDoS attack. The attacker is assumed to know the shape of the source address distribution in the legitimate traffic, but not the actual IP addresses in the frequency ranking.

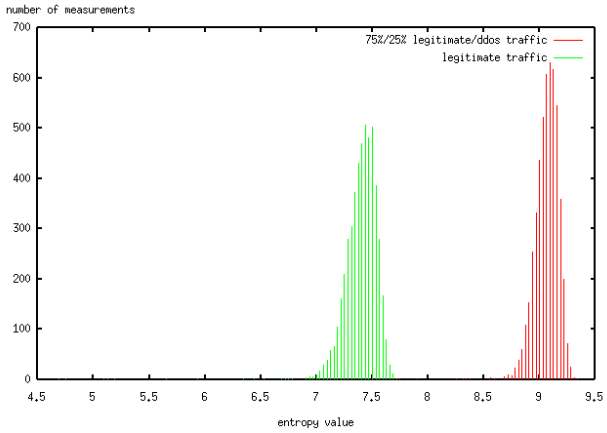

Figure 4: Distribution of source address entropy under normal and typical-DDoS attack conditions

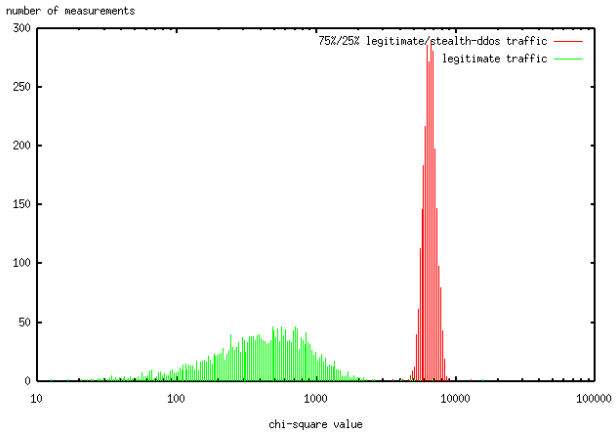

Figure 5: Distribution of source address chi-square values under normal and typical-DDoS attack conditions

The simulated attack traffic thus has the same source-address frequency distribution as the legitimate traffic, but uses a different set of source addresses. There is now significant overlap between the chi-square values observed under normal conditions and under attack, suggesting that, depending on the threshold chosen, the chi-square detector would either frequently fail to detect such attacks or issue a large number of false-positive alerts.

Under the same stealthy attack, the entropy detector (Figure 6) fares somewhat better, but a different form of stealthy attack would be more effective against entropy detection. Specifically, if the attacker knows approximate values for (1) the average source address

entropy value observed by the detector and (2) the fraction of traffic seen by the detector that will be attack traffic, then he or she can emit attack traffic with an entropy somewhat lower than the average value to compensate for the entropy increase resulting from the use of a disjoint sets of source addresses. This way, an attacker armed with this knowledge of the detector environment could produce attack traffic that would produce little change in the entropy observed at the detector.

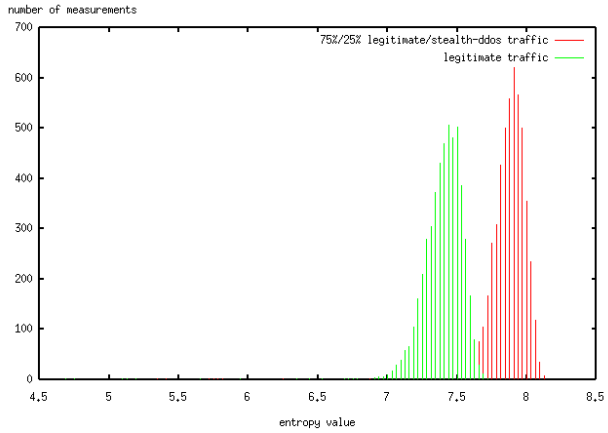

Figure 6: Distribution of source address entropy under normal and stealthy-DDoS attack conditions

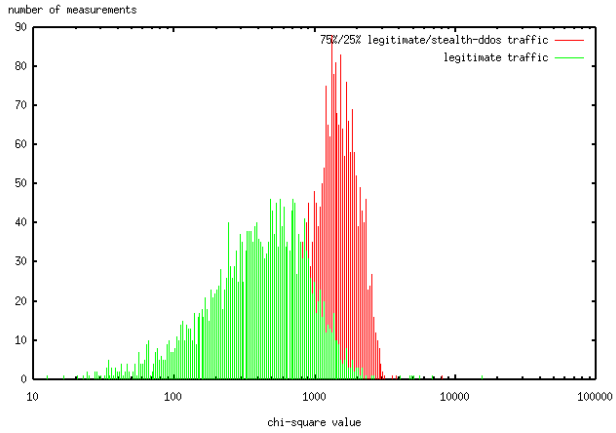

Figure 7: Distribution of chi-square values for source address under normal and stealthy-DDoS attack conditions

Tables 1 and 2 show the results of running similar experiments on the different traffic traces described in Section 3.2, with a variety of attack and detector combinations. Each entry in the table represents the fraction of DDoS measurements that fall above the 95th percentile for legitimate traffic (for entropy detectors, the fraction that fall below the 2.5th percentile or above the 97.5th). Values close to one suggest that effective attack detection is possible. The "random," "fixed," and "sequential" attack characteristics are typical of current DDoS attack tools; the "stealth" characteristics represent potential future attack tools using the approach described above.

These tables indicate at least two noteworthy effects. First, random source address and sequential destination port attacks are easily detected in most environments. Legitimate traffic in almost any network contains relatively small sets of addresses and ports that constitute a substantial fraction of traffic. Hence, uniform distributions will usually stand out. Second, results are consistently better for the NZIX trace, which is the most representative of core Internet traffic, our target environment. The other data sets, especially "Small Company," exhibit substantially more variability in legitimate traffic patterns, so attacks are harder to distinguish from legitimate fluctuations. These traces are taken from "edge" networks in which during quieter periods, traffic is frequently dominated by a handful or even a single pair of hosts. To avoid false positives from these harmless anomalies while preserving sensitivity to attack, detectors can be modified to suppress alarms when overall throughput is low. Some low-rate attacks may be missed with this approach, but the high-volume floods should still be detected.

Table 1: Entropy detector accuracy for current and future attack types

	data set			
attack characteristic	NZIX	University	Bell Labs	Small Co.
random src addr	0.999	0.850	0.920	0.544
stealth src addr	0.982	0.283	0.475	0.339
fixed dest addr	1.000	0.012	0.005	0.000
sequential dest port	1.000	0.940	0.857	0.949
stealth dst port	0.987	0.328	0.340	0.606

Table 2: Chi-square detector accuracy for current and future attack types

	data set			
attack characteristic	NZIX	University	Bell Labs	Small Co.
random src addr	1.000	1.000	1.000	0.999
stealth src addr	0.540	0.159	0.253	0.000
fixed dest addr	1.000	0.076	0.074	0.045
sequential dest port	1.000	1.000	0.999	0.247
stealth dest port	0.970	0.466	0.030	0.096

We contend that the prospects for detecting stealthy attacks are not as bleak as they might appear, for several reasons. There are many situations in which, even if the attackers had full knowledge of the detector's network environment, their ability to emulate the typical traffic mix and "blend in" would be limited. Some of the most powerful DDoS attacks do not use completely compromised machines as the flood

sources, but rather exploit "reflectors" that can be caused to respond to and amplify particular kinds of traffic. In such situations, the attacker does not have full control over the *reflected* packets' form; for instance source address spoofing is typically impossible. In addition, attack traffic may have to use certain ports, flags, addresses, and other characteristics to achieve its primary goal or to evade other countermeasures such as firewalls and egress filtering. These and other constraints on the attackers may cause the attack traffic to exhibit atypical characteristics at one or more points along the path to the victims, so detectors at those points may be able to identify the attack.

3.5. Detector Performance

Since we are proposing to use these detection methods in high-speed core routers, it is imperative that they have low computational cost, especially for the operations that must be carried out for each packet. The prototype Snort detector implementation exhibits adequate performance for its purposes: on a 1GHz Pentium-III-based machine, a Snort process running a single chi-square detector observing source addresses can process 240,000-270,000 packets per second (pps) offline. (The Snort infrastructure without any plugins can handle 435,000 pps.) Adding chi-square detectors for four additional packet attributes brings performance down to around 100,000 pps. A single-attribute entropy detector can manage about 294,000 pps, while adding six others yields 130,000 pps. These speeds are roughly in the OC3 range. Improving performance is a primary goal of future detector development. We expect to achieve improved performance by implementing some optimizations that approximate the true frequency profile while reducing or eliminating floating-point operations in the packet-handling code. Most of the partitioning and computation of chi-square and entropy values can be handled asynchronously in background processes that should not impede the packet-handling fast path.

4. Response

Our defense approach involves *response modules* that use a characterization of the attack provided by the detection module to take defensive measures. The response module classifies individual packets as benign or suspect based on the attack characteristics provided by the detector. Once identified, the suspect packets are subjected to rate limiting or packet-filtering methods based on the intensity of the attack or pre-defined response policies. In the case of stealthy DDoS attacks, the response module should communicate with the detector and share the data structures and statistical models maintained by the detector to identify the attack packets with high confidence; the prototype described below does not yet offer such coordination.

4.1. Prototype DDoS Response Module

The current response prototype is implemented on a Linux router as a kernel module. It uses netfilter and Linux Advanced Routing and Traffic Control (LARTC) to filter and rate-limit packets [15],[4]. An API is provided to take alerts from the detection module and generate filter rules to be issued to the response module. We have also produced an extension to the Linux *iptables* mechanism that provides similar functionality, for better integration with iptables-based router/firewall configurations.

Currently, the response module implements three packet-filtering rules. They are *constant*, *random* and *allow*. These filter rules are automatically generated by the Snort-based DDoS detector when it issues an alert.

When the detector module detects a DDoS flood, it uses its detection algorithms and the statistical models to characterize the DDoS packets. The characteristics of the DDoS packets are used to form one of the three packet filter rules. The detector can then insert the packet filtering rules using the /proc file system interface. These rules will filter out the DDoS attack packets. Once the detector determines that the attack has subsided, it can remove the appropriate filtering rules.

4.1.1. *Constant* Filter Rule. A *constant* filter rule is used to drop packets that match the values specified in the rule to the values of the protocol header fields in the packet. This filter rule can be applied to the IP header fields, TCP source and destination ports, UDP source and destination ports and ICMP type and code fields. For example, the rule *"const {daddr=10.1.1.10 protocol=6 dport=80 sport=31137};"* will drop TCP packets going to the particular destination IP address 10.1.1.10 and TCP destination port 80 from the TCP source port 31137. A number of such *constant* filter rules can be applied to the response module. If none of the *constant* filter rules match the values of the header fields, the packet is allowed to pass.

4.1.2. *Random* Filter Rule. A number of DDoS attack tools create packets with random values in the header fields. That is, a random number generator is used to assign a value to certain fields in the header. In such a case, a *random* filter rule can be specified to drop packets with random values in the header field. The *random* filter rule can be applied to the IP header fields, TCP source and destination ports, UDP source and destination ports, and ICMP type and code fields.

Currently, a simple algorithm is used to determine if a packet has a random value for a particular header field. When a *random* filter rule is invoked for a header field (for example source IP address), the count of each distinct source IP address seen at the detector is recorded. The count is started from the instant that particular filter rule is applied to the response module. If the count is less than a pre-determined threshold value, it is assumed that a random number generator is setting the source IP address and the packet is dropped. If the count is more than the threshold value then the packet is allowed to pass. The count is reset to zero when the random filter rule is removed.

Random filter rules can be applied in the following cases:

- A packet has only one random item in the header field. For example, the rule *"rand {saddr};"* will drop packets with only random source IP addresses.
- A packet with more than one random item in the header field. For example, the rule *"rand {tot_len saddr protocol};"* will drop packets with random values for total IP packet length, source IP address and IP protocol.

Since this simple approach allows all packets with a given value to pass after the threshold is reached for that value, an attacker could choose a distribution of attack packets that limits the filter's effectiveness. For example, if the attack packets have source addresses drawn at random from a set of 100 addresses, the filter will drop the first few packets with each source, but permit the rest. However, by limiting the range of values used, the attacker loses part of the benefit of using randomly chosen values; in particular, it becomes easier to detect and block the attack by other means. Another problem with this approach is that in some network environments, dropping the first few packets with (for example) a previously unseen source address may have an unacceptable impact on legitimate traffic. Filtering on multiple packet attributes simultaneously and replacing the threshold with a graduated quality-of-service scheme could mitigate this drawback.

4.1.3. *Allow* Filter Rule. During a DDoS attack, there may be a need to allow a particular kind of traffic to pass through, such as the communication traffic between routers or between routers and command centers or between critical applications. The *allow* packet rule is used to allow packets to pass that match the values specified in the rule to the values of the protocol header fields in the packet. The *allow* filter rule can be applied to IP header fields, TCP source and destination ports, UDP source and destination ports, and ICMP type and code fields. For example, the rule *"allow {daddr=10.1.1.10 saddr=10.60.33.1 protocol=6 dport=80};"* will allow TCP packets going to destination IP address 10.1.1.10 and TCP destination port 80 from source IP address 10.60.33.1 to pass. A number of such *allow* filter rules can be applied to the response module to permit regularly occurring traffic to pass.

The use of the *allow* rule can also be envisioned in the case where the overhead of adding the random or constant rules to block a large sub-set of the traffic (i.e. DDoS traffic) is far greater than the overhead of adding allow rules to allow a smaller sub-set of the traffic (i.e. good traffic).

4.1.4. `/proc` File System Interface. A `/proc` file system interface is provided for the detector to communicate with the response module. The interface allows the detector to configure the response module with different packet filter rules. The interface provides two options to configure the response module: *insert* and *clear*.

The *insert* option is used to insert/add new packet filter rules. A rule number and one of the three filter rules discussed above must be specified. The *insert* rules are evaluated in the following order of precedence: *constant, random,* and *allow*. . Within each of the filter types, the rules are evaluated in the ascending order of the rule numbers.

The *clear* option is used to remove filter rules. The filter type in conjunction with the rule number or header field is required as a parameter to the *clear* option.

4.2. Extending detectors to recommend response

Both chi-square and entropy DDoS detectors can be extended to provide attack characterization information that can be used to target packet-filtering or rate-limiting responses to mitigate the effects of DDoS attacks.

4.2.1. Chi-Square Detector. In chi-square detectors, attack detection is triggered by an unusually high chi-square statistic. That unusually high value, in turn, must result from one or more bins whose frequency differs substantially from the baseline. In order to determine the most anomalous bin, the detector need only find the largest terms in the chi-square sum. A reasonable approximate attack characterization can be constructed by simply identifying the most anomalous bin whose frequency is too high. For example, if the distribution of source addresses is unusually dispersed, the last bin will have a high frequency, so an appropriate response would be to rate-limit traffic belonging to that bin. This is not an exact characterization and some legitimate traffic will be adversely affected by the response, but

the legitimate traffic belonging to other bins will be unaffected by the rate limiting and should benefit from the reduced flood traffic.

One minor problem with this approach is that the assignment of values to bins normally changes at each new chi-square computation, since the assignment is based on sorted frequency. However, the response can use the most recent bin assignment to classify packets. Assuming re-sorting is done at intervals comparable to the current profile decay half-life this should provide a good approximation.

One direction for future work is to correlate information about different packet attributes in order to more narrowly target the response. The more precisely the attack traffic can be characterized, the smaller the "collateral damage" done by the response rate limiting will be. Furthermore, by looking simultaneously at multiple attributes, detectors may achieve greater accuracy. Traffic that seems borderline anomalous or even typical when different attributes are examined in isolation may stand out clearly when the combined distribution of different attributes is considered. For example, a high rate of connection attempts on TCP port 80 with the destination address of a large web server may be normal, while the same rate of port-80 attempts targeting a DNS server may indicate an attack. Monitoring multiple packet attributes simultaneously in a way that detects such anomalies without requiring excessive memory usage by the detector could yield significant benefits for both detection and response.

4.2.2. Entropy Detector. If the entropy detector determines that the current entropy for some attribute is below the normal range, that suggests that traffic with a relatively small number of values for that attribute is dominating. Since the entropy detector tracks value frequency, it can identify which values are the most common and are likely candidates for rate limiting. For finer targeting, the detector could watch for specific values with dramatic increases in frequency and treat those as suspicious.

Conversely, an unusually high entropy value suggests that the low-frequency values are causing trouble, so the detector might suggest that packets having high-frequency values be given preferential treatment.

4.2.3. Integrating detection and response. We have implemented two detection/response integration mechanisms.

First, we have built a Snort alerting module that can issue alerts using the Intruder Detection and Isolation Protocol [16]. By modifying the Snort alert model to be more extensible, we enabled the Snort-based chi-square and entropy detectors to communicate additional information on attack characteristics to the IDIP alerting module for reporting to a remote response module closer to the attacker.

Second, we modified the Snort-based chi-square and entropy detectors to issue rate-limiting directives to the iptables-based response module described in Section 4.1.

Both of these integration approaches impose fairly strict limits on the amount of information that can be exchanged between the detector and the responder. This means that, while a chi-square detector might ideally instruct the responder to rate-limit all traffic in a given bin, it must instead approximate this order by providing a small number of values (e.g., IP addresses) to be rate-limited, since only the detector knows all the IP addresses belonging to the bin. With a more tightly integrated pair of detection and response modules, the responder could query the detector for each new IP address it sees, in order to determine whether to apply a rate limit for that address. This approach would allow response decisions to take full advantage of the information already collected by the detector. We plan to implement such an approach using a netfilter-based kernel module with access to the address space of a user-level detection process, as described in Section 5.

4.3. DDoS Response Module Evaluation

The current response prototype is an initial implementation of the response system. Initial experimental results have indicated that the response prototype blocks substantial DDoS attack traffic generated by the Stacheldraht attack tool. The Stacheldraht attack tool generates DDoS attacks with constant packet attributes. Though this is not an evidence for actual effectiveness of the response system, it is a promising step.

The *constant* rule implemented by the current prototype is a case of an extreme response method, which is to block or drop all the traffic. The *random* rule has the basic drawback of dropping the first few packets of every new good connection. These two rules could potentially increase the false negatives. The *allow* rule could allow through some of the DDoS attack traffic that matches the rule, increasing the false positives.

Further experimentation is planned to determine the effectiveness of the response system and also to determine the rate of false positives and false negatives.

5. Summary and Future Extensions

The focus thus far has been on detection and response algorithms and the implementation of these algorithms in software. At issue is whether these algorithms can reliably detect and respond to DDoS attacks.

Against today's relatively unsophisticated DDoS toolkits, our prototype detector is able to determine that the network is under attack and deploy accurate filtering rules. The filtering effort is immediate and reduces the impact of the attack downstream almost instantly. Because baseline measurements and thresholds can be established automatically, and because detectors can generate filtering rules automatically based on the traffic statistics they gather, the system is adaptable to a wide range of network environments with minimal manual tuning. While our initial goal was to provide effective defense against existing DDoS tools, we are continuing to explore techniques for better defense against future stealthy attacks.

Future research and development will focus on tighter integration of detection and response modules. In the current implementation, detectors generate concise recommended rules for responders to impose, and there is no further detector/responder coordination. In a more tightly coupled detection/response system, the individual packet classification decisions made by the responder could make use of the rich data structures maintained by the detector. This would enable more focused filtering and rate limiting, and reduce the possible impact of responses on legitimate traffic.

Another approach to providing more narrowly targeted response while avoiding computationally expensive analysis would be to enable detectors to dynamically tune themselves and "drill down" to investigate detected anomalies more closely. A detector with these capabilities could more effectively allocate its limited computational resources where they are most needed. Such drill-down could be triggered by a vague or uncertain detection by a quick analysis, or by complaints received from downstream network devices.

The Linux implementation of this system has been appropriate for demonstration environments and evaluation of alternative detection approaches. The next step is to port this prototype system to the Intel IXP-1200 network processor. We consider this processor representative of the next generation of network hardware in that it is a highly programmable device with the capability of forwarding network traffic at high bandwidth. By implementing detection and response methods on this platform and testing their performance, we can validate the claim that they are appropriate for use in future high-speed routers.

6. Acknowledgements

This research is supported by grants from DARPA. Our experiments have benefited from a large supply of internet traffic data provided by the Measurement and Network Analysis Group at the National Laboratory for Applied Network Research (NLANR), Waikato Applied Network Dynamics research group at the University of Waikato Computer Science Department, Lincoln Laboratories and the Computer Science Department at UCLA. We also thank Dr. Brett Tjaden and Dr. Shawn Ostermann for providing network trace data from the Engineering and Technology network at Ohio University.

7. References

[1] D. Dittrich, "The 'Stacheldraht' Distributed Denial of Service Attack Tool", http://staff.washington.edu/dittrich/misc/stacheldraht.analysis, 1999.

[2] C. Faloutsos, M. Faloutsos, and P. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. of ACM SIGCOMM*, Aug. 1999, pp. 251-262.

[3] T.M. Gil, M.A. Poletto and E.W. Kohler, Jr. "Statistics Collection for Network Traffic", United States Patent Application, March 21, 2002.

[4] B. Hubert, "Linux Advanced Routing and Traffic Control HOWTO", http://lartc.org/howto/.

[5] D. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Third edition, Vol. 2, Addison-Wesley, Reading, Massachusetts, 1997.

[6] M.V. Mahoney and P.K. Chan, "Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks", *SIGKDD '02*, Edmonton, Alberta, Canada, July 23-26, 2002, pp. 376-385.

[7] R. Manajan, et al., "Controlling High Bandwidth Aggregates in the Network", *SIGCOMM Computer Communications Review*, 32(3), July 2002.

[8] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity", *Proceedings of USENIX Security Symposium 2001*, pp. 9-22.

[9] E. Mouw, "Linux Kernel Procfs Guide", http://www.kernelnewbies.org/documents/kdoc/procfs-guide/lkprocfsguide.html.

[10] Netflood Infosec Tools & Resources, Source Code to Stacheldraht, http://netflood.net/files/Dos/DDoS.

[11] O. Pomerantz, "Linux Kernel Module Programming Guide", http://www.tldp.org/LDP/lkmpg/mpg.html.

[12] P.A. Porras, and P.G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *Proceedings of the National Information Systems Security Conference (NISSC)*, October 1997, pp. 353-365.

[13] M. Roesch, (March 2002), "Snort Users Manual: Snort Release 1.8.5", http://www.snort.org/documentation.html, March 2002.

[14] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks" *Proceedings of the 13th Systems Administration Conference (LISA'99*), USENIX Association, 1999, pp. 229-238, http://www.snort.org/docs/lisapaper.txt.

[15] R. Russell and H. Welte, “Linux Netfilter Hacking HOWTO”, http://cvs.netfilter.org/cgi-bin/cvsweb/netfilter/documentation/HOWTO/.

[16] D. Schnackenberg, K. Djahandari, and D. Sterne, “Infrastructure for Intrusion Detection and Response”, *DISCEX 2000*, January 2000, pp. 1003-1011.

[17] C.E. Shannon, and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1963.