A Source Router Approach to DDoS Defense

Jelena Mirkovic, Peter Reiher and Gregory Prier
Computer Science Department
University of California, Los Angeles

Abstract

Distributed denial-of-service attacks present a great threat to the Internet, and existing security mechanisms cannot detect or stop them successfully. The problem lies in the distributed nature of the attacks, which engages the power of a vast number of coordinated hosts. The response to the attack needs to be distributed also, but cooperation between administrative domains is hard to achieve, and security and authentication of participants incur high cost. We propose a DDoS defense system deployed at source-end networks that autonomously detects and stops the attacks originating from those networks. Attacks are detected by monitoring two-way traffic flows between the network and the rest of the Internet. Monitored flows are periodically compared with predefined models of normal traffic, and those flows classified as part of DDoS attack are rate-limited. We evaluate the performance of our system in a realistic testbed.

1. Introduction

Distributed denial-of-service (DDoS) attacks are comprised of large numbers of packet streams from disparate sources. These streams converge on the victim consuming some key resource and rendering it unavailable to legitimate clients. This can be done by exploiting vulnerabilities in systems or protocols at the victim network, or by merely requesting normal services at intense rates.

There are no common characteristics of DDoS streams that could be used for their detection and filtering. The attacks achieve their desired effect by sheer volume of the attack packets, and can afford to vary all packet fields to avoid characterization. In addition to this, the attackers follow advances in the security field and adjust their tools to defeat new security systems.

The cooperation of distributed sources makes DDoS attacks hard to combat or trace back. Any defense against these attacks must address their distributed nature to be successful. Several distributed DDoS defense systems have been proposed that deploy cooperation between intermediate routers to combat an attack. These routers are augmented to monitor traffic and grant requests for rate-limiting of the streams they deliver to their peers. While these approaches seem promising, they require the cooperation of every router from the source to the destination host for response propagation. Since the Internet is not under any single administrative control, global deployment of such mechanisms is hard to enforce. Also, cooperation between routers requires strongly secured and authenticated communication, which incurs high cost.

The attacks ideally should be stopped as close to the sources as possible, saving network resources and reducing congestion. In this paper we propose a DDoS defense system that is deployed at the source-end and prevents the machines at associated network from participating in DDoS attacks. This system is deployed at the network's "exit" router and monitors two-way traffic between the network and the rest of the Internet. The online traffic statistics are compared to predefined models of normal traffic, and non-complying flows are rate-limited. Section 2 gives a general overview of DDoS attacks. Section 3 describes the proposed DDoS defense system, and Section 4 gives more details about the system's architecture and explains the functioning of specific components. Section 5 presents experimental results that illustrate the performance of the proposed system. Section 6 gives an overview of related work. Section 7 investigates security issues, and Section 8 discusses implementation considerations such as partial deployment. Section 9 discusses open problems and future work directions, and Section 10 concludes the paper.

2. DDoS Attacks

Distributed denial-of-service attacks exploit different strategies to exhaust the resources of the victim. Some protocol implementations have bugs that allow a few malformed packets to severely degrade server or network performance. The victim can frequently prevent these attacks by implementing various patches to fix the vulnerability, or by filtering malformed packets. On the other hand, DDoS attacks can leverage the power of many distributed machines to make a massive number of legitimate requests for a service from a single host. Since these requests are legitimate, the victim cannot refuse to service them, nor can it recognize them as part of the attack until its resources are exhausted.

Attackers follow trends in the network security field and adjust their attacks to defeat current defense mechanisms. Spoofing of source addresses is used to avoid traceback and decoy packets and encryption are used to combat signature-based detection. We now provide a quick overview of the several well-known DDoS attack tools in order to illustrate the variety of mechanisms deployed.

Trinoo [25] is a simple tool used to launch coordinated UDP flood attacks against one or many IP addresses. The attack uses constant-size UDP packets to target random ports on the victim machine. The master uses UDP or TCP to communicate with the slaves. This channel can be encrypted and password protected as well. Trinoo does not spoof source addresses although it can easily be extended to include this capability.

Tribe Flood Network (TFN) [26] can generate UDP and ICMP echo request floods, TCP SYN floods and ICMP directed broadcast (e.g. Smurf). It can spoof source IP addresses and also randomize the target ports. Communication between masters and agents occurs exclusively through `ICMP_ECHO_REPLY` packets.

Stacheldraht [27] combines features of Trinoo (master/agent architecture) with those of the original TFN (ICMP/TCP/UDP flood and "Smurf" style attacks). It adds encryption to the communication channels between the attacker and Stacheldraht masters. Communication is performed through TCP and ICMP packets. It allows automated update of the agents using `rcp` and a stolen account at some site as a cache. New program versions will have more features and different signatures to avoid detection.

TFN2K [15] is the variant of TFN that includes features designed specifically to make TFN2K traffic difficult to recognize and filter. Targets are attacked via UDP, TCP SYN, `ICMP_ECHO` flood or Smurf attack, and the attack type can be varied during the attack. Commands are sent from the master to the agent via TCP, UDP, ICMP, or all three at random. The command packets may be interspersed with any number of decoy packets sent to random IP addresses to avoid detection. In networks that employ ingress filtering as described in [24], TFN2K can forge packets that appear to come from neighboring machines. All communication between masters and agents is encrypted and base-64 encoded.

The **mstream** [28] tool uses spoofed TCP packets with the ACK flag set to attack the target. Communication is not encrypted and is performed through TCP and UDP packets. Access to the master is password protected. This program has a feature not found in other DDoS tools. It informs all connected users of access, successful or not, to the handler(s) by competing parties.

Shaft [9] uses TCP, ICMP or UDP flood to perform the attack and it can deploy all three styles simultaneously. UDP is used for communication between masters and agents and messages are not encrypted. Shaft randomizes the source IP address and the source port in packets. The size of packets remains fixed during the attack. A new feature is the ability to switch the master's IP address and port during the attack.

The **Code Red** [13] worm is self-propagating malicious code that exploits a known vulnerability in Microsoft IIS servers for propagation. It achieves synchronized attack by preprogramming the onset and abort time of the attack, attack method and target addresses (i.e. no master/agent architecture is involved).

In addition to sophisticated attack tools, attackers also use a set of automated scripts for self-propagation of malicious code and different techniques to mask the code at the source machine.

3. Source Router Approach

Placing DDoS defense close to the sources of the attack has many advantages over placing it further downstream. The attack flows can be stopped before they enter the Internet core and before they aggregate with other attack flows, and achieve the power to create network congestion and exhaust resources of the victim and intermediate routers. Being close to the sources can facilitate easier traceback and investigation of the attack. Due to the low degree of flow aggregation, more complex detection strategies can be deployed to achieve higher accuracy. Also, routers closer to the sources are likely to relay less traffic than core routers and can dedicate more of their resources to DDoS defense at a lower performance impact.

The *Source Router Approach* uses the router (hereafter called the *source router)*, which serves as a gateway between the *source network* containing some of the attack nodes and the rest of the Internet, to detect and limit DDoS streams long before they reach the target. The design proposed in this paper assumes that the source

network is an edge network, i.e. most of the traffic passing through the edge routers either originates within this network or is destined for some machines in this network. We assume that the source router is able to identify the interfaces on which it receives the source network's incoming and outgoing traffic, either through some protocol or through manual configuration. We further assume that all machines on the source network use this router as the "exit router" to reach a particular set of destinations. The source router is thus in a unique position to protect these destinations from denial-of-service attacks originating at the source network by arbitrating communication with them.

As described in Section 2, typical denial-of-service flows lack common characteristics that would enable us to detect and filter them by traditional methods. These attacks are therefore usually detected after their effect is felt by the victim, or after the heavy aggregation of attack flows creates congestion somewhere in the network. Detection of denial-of-service attacks is particularly hard at the source end, since the attacking flows might not be heavily aggregated at this point in the network, and the effect of the attack is usually not so prominent.

The source router can detect the attack by observing two-way communication between machines on the source network and the outside hosts. The source router monitors the behavior of each destination with which the source network communicates, looking for the difficulties in communication, such as reduction of number of response packets or longer inter-arrival times. These difficulties can be a sign of a denial-of-service attack against the specific destination or of severe network congestion on the route to the destination. These difficulties are detected by periodically comparing the parameter values of the two-way traffic for each destination against a predefined model of normal traffic. If the comparison reveals the possibility of DDoS attack, the source router responds by imposing a rate limit on all outgoing traffic flows for this destination. The outcome of subsequent observation intervals either confirms or refutes this hypothesis. Confirmation further restricts the allowed rate limit, whereas refutation leads to a slow increase of the allowed outgoing traffic rate, according to the degree of well-behavior of outgoing flows.

The source router detects the attack and responds to it autonomously, without communication with other routers or human intervention. More accurate attack detection might be achieved if the source router relied on a signal from the victim instead of its own incomplete observations. However, this approach requires reliable, secure and authenticated communication between the victim and the source router, which cannot be guaranteed. Also, a more complete observation and more appropriate responses might be possible if all border routers in the source network were allowed to exchange information. Since security and authentication might be easier to achieve for those machines, this approach seems promising and is part of our future work.

4. System Architecture

The DDoS defense system proposed in this paper consists of *observation* and *throttling* components, which can be part of the source router itself or can belong to a separate unit that interacts with the source router to obtain traffic statistics and install filtering rules. Figure 1 depicts the architecture corresponding to the second approach. The observation component monitors the outgoing and incoming packets and gathers statistics on two-way communication between the source network and the rest of the Internet. These statistics are classified per destination. Periodically, statistics are compared to a predefined model of normal traffic, and the result of this comparison is passed to the throttling component. The throttling component adjusts the filtering rules based on this characterization and on flow behavior.

A prototype of the system incorporating all the elements described here has been built and tested. Section 5 describes the results of those experiments.

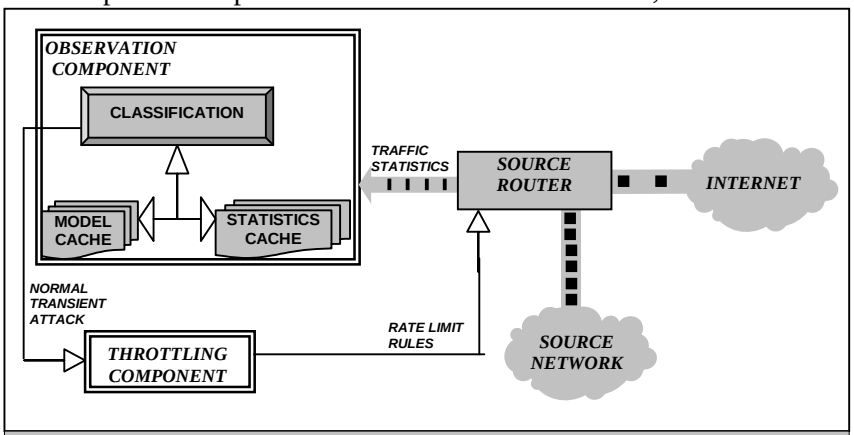

Figure 1: Architecture of the proposed DDoS defense system

4.1. Observation Component

The observation component monitors all the traffic passing through the source router. Each packet is classified as *incoming* or *outgoing* based on its source or destination address, or arriving interface.[1] Information in the packet header is then used to update statistics on current flows. Periodically, statistics are compared with a model of normal traffic, and flows are characterized as being *normal*, *transient* or *attack* flows. Normal flows are those whose parameters match those of the model and who have not been recently classified as attack flows. Attack flows are those whose parameters are outside of the model boundaries. Transient flows are those whose parameters match those of the model but which have been recently classified as attack flows; thus their rate-limit must be relaxed slowly to avoid recurring attacks.

4.1.1. Statistics Gathering

The purpose of statistics gathering is to characterize the current behavior of each Internet host with which the source network is communicating. The statistics provide the information needed to discover difficulties in communication caused by denial-of-service attacks on the destination host, or by network congestion along the route to the destination. The statistics are classified per IP address of *foreign*[2] Internet host. Keeping a record for each existing IP address is infeasible, so the size of the statistics cache is limited and the cache is purged periodically. Figure 2 shows the number of cache records in subsequent observation intervals obtained by gathering statistics from different traces: (1) a trace collected at the "exit" router of the UCLA Computer Science Department network (2) a LBL-PKT trace from Lawrence Berkeley Laboratory [20], (3) a DEC-PKT trace from Digital Equipment Corporation [21], and (4) a NLANR-auckland trace from [22]. Even though the cache size varies among different traces, it remains fairly stable across time for a particular trace. This justifies our assumption that the maximum cache size can be preset for a given source network and thus the storage requirement can be limited while not losing important information. The cache is purged after flows are categorized by deleting the records for flows that have been classified as normal during the comparison. Records for the attack and transient flows are kept, but the statistics are reset. If the cache overflows during the observation interval, the record for the destination receiving the smallest number of bytes is expunged.

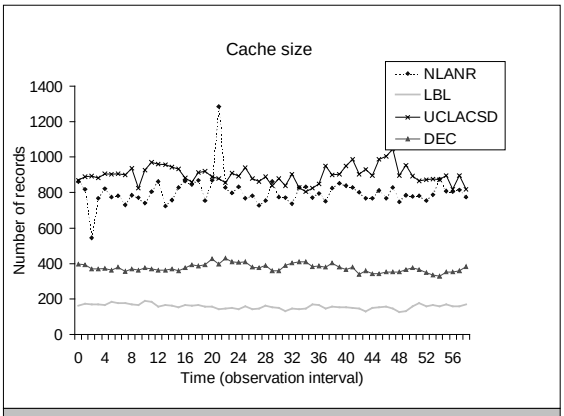

Figure 2: Cache size for several traces

Only data from the packet's IP header is used for statistics gathering. This allows fast operation since most routers are optimized for fast lookup of IP header fields and classification of packets based on these fields. This choice also reduces the size of the state kept for each record. Finally, it provides the possibility of verification of our detection technique against many published packet traces.

A record in the statistics cache corresponding to some peer address PA consists of the following fields:

p_{sent} - number of packets sent to PA
p_{rec} - number of packets received from PA
B_{sent} - number of bytes sent to PA
B_{drop} - number of bytes dropped due to rate limit
int_{sent} – inter-arrival time of packets sent to PA
int_{rec} – inter-arrival time of packets received from PA
mean_rto – smoothed mean of the ratio of number of packets sent to and received from PA
timestamp – timestamp when fields were last reset

[1] The exact classification method depends on how the router is connected to the source network. It is obvious that classification by source address cannot be performed reliably if spoofing is possible. If a router does not relay transit traffic, or it can clearly identify "incoming" and "outgoing" interfaces with regard to its source network, then classification by destination address or arriving interface can be performed. In our testbed we used classification by destination address.

[2] *Foreign* in this context means the host that does not belong to the source network.

4.1.2. Traffic Models

Most of communications between Internet hosts involve two-way traffic. The TCP protocol uses a two-way communication paradigm to achieve reliable delivery. During a TCP session, the data flow from source to destination host is controlled by the constant flow of acknowledgments in the reverse direction. Thus, normal TCP communication is modeled by small values of the ratio of number of packets sent to a specific destination and number of packets received from this destination. Ideally, this ratio should be one, but network congestion and different TCP implementations using delayed and selective acknowledgements push it to slightly larger values.

Other communication protocols (UDP, ICMP) do not explicitly require reverse traffic for proper operation. Some applications that use these protocols expect responses from the peer host. The *ping* application sends out `ICMP_ECHO` packets and expects to receive `ICMP_ECHO_RESPONSE` from the peer host. *ICQ* and *NetMeeting* use UDP packets to convey text, audio and video between humans; during normal operation messages travel in both directions. *DNS* requests carried in UDP packets usually produce a DNS reply in the reverse direction. For non-TCP traffic, servers usually experience smaller amounts of reverse traffic than clients, since their service consists of streaming real-time data and occasionally receiving control packets or streaming requests. However, with regard to a specific destination, those servers send a fairly stable volume of traffic per time interval, and we use this feature to characterize normal non-TCP traffic.

Models of normal non-TCP traffic are created during the *training phase* before the DDoS defense system is started. During this phase the observation component gathers statistical records whose structure is similar to those gathered during the *attack detection phase*, but not necessarily in consecutive observation intervals. Traffic can be sampled instead of constantly monitored, and the observation interval need not be the same length as the observation interval during the attack detection phase. In addition to per-destination classification, in this phase traffic is also profiled per type, and records are kept in separate caches describing different types of non-TCP traffic. In our experiments, we differentiated between four types of traffic: TCP, UDP, ICMP and *other*. Each incoming packet is classified per type and destination and a corresponding cache record is updated.

At the end of each interval, the cumulative traffic parameters (mean and standard deviation) for a specific destination PA are updated based on the data gathered during the interval. After the update, the cache is purged, and the statistics gathering process is restarted at a later time.

After the training phase is finished, a destination is classified as TCP, UDP, ICMP or other based on the traffic type of the majority of packets sent to this destination. If the dominant type of traffic is non-TCP, the cumulative parameters corresponding to it are stored in a file. This file is used to initialize models of non-TCP traffic prior to the recognition phase. Destinations that have no dominant type of traffic (i.e. no type of traffic is represented by more than 50% of the packets sent) are conservatively classified as TCP destinations. Records that have a small number of observations are discarded as statistically insignificant, and the corresponding destinations are treated as TCP destinations.

Classification of destinations as strictly one type (e.g. TCP or UDP but not both) will introduce errors in the detection process. It would be more appropriate to use the complete profile of the destination to detect the attacks, but this would consume four times more memory than the proposed approach, and would require online classification of incoming packets per traffic type and four times larger online statistics.

Although the traffic models are generated prior to system startup, they need to be updated periodically to reflect new trends in network traffic, such as the deployment of a new service. The update of models needs to be controlled to prevent attackers from training the system over time to regard the attacks as normal traffic. The automatic update of models of normal non-TCP traffic is part of our future work.

4.1.3. Classification of Traffic Flows

The purpose of the flow classification is to detect if the associated destination experiences communication difficulties that could be a sign of a DDoS attack. During classification, flows are first matched with the models of normal traffic and classified as *compliant* or attack flows. Compliant flows are further examined and classified as either normal or transient flows based on their behavior.

The gathered flow statistics are the aggregate statistics for both TCP and non-TCP traffic in the flow. As a first classification step these statistics are compared with the more restrictive model of normal TCP traffic. The smoothed ratio of number of packets sent and number of packets received for a flow destination is compared with the maximum allowed packet ratio for TCP traffic. Smoothing is performed by exponentially averaging the sampled values of this parameter to accommodate temporary peaks. If the smoothed packet ratio for the specific destination is smaller than maximum allowed, the flow is classified as compliant. Otherwise, further classification is needed using the less restrictive models of normal non-TCP traffic. If the model exists for the corresponding destination, and observed parameter values are within limits defined by the model, the flow is classified as compliant. Otherwise, if no model exists or observed parameter values are outside limits defined by the model, the flow is classified as attack.

Flows that are classified as compliant are further checked to see if they are well behaved.[3] To facilitate this check, a *well-behaved* counter is associated with each cache record. The purpose of this counter is to slow down the recovery of flows that have been classified as attack flows and keep their records in the cache long enough to assure they are well behaved. The counter is originally unset, and can be modified after the flow has been matched with the model of normal traffic. The counter is set to 0 every time a flow is classified as attack; and incremented, if it is set, every time the flow is classified as compliant. Compliant flows that have an unset value of this counter are classified as normal, otherwise they are classified as transient. The value of the *well-behaved* counter indicates the number of consecutive observation intervals when the flow has been punished and behaved well. The counter is unset after the flow has endured an appropriate penalty.

4.2. Throttling Component

As a response to the attack, the throttling component restricts the allowed sending rate to the victim of the attack. Since the detection of the attack is unreliable and may have false-positives, rate-limiting is a better-suited response than complete filtering. Filtering out all the traffic to the victim would greatly damage misclassified flows, whereas rate-limiting still allows some packets to reach the destination and thus keeps connections alive. Allowing some attack packets through is acceptable, since the attack's overall impact depends on the volume of the attack packets.

The throttling component defines the allowed sending rate to a particular destination based on the history of its behavior and current classification of its associated flow. The detection of the attack is based on incomplete observations and is not reliable. Therefore, the throttling component has the hard task of defining a strategy that should be able to effectively limit attacking flows and reduce the effect of the attack on the victim, while at the same allowing misclassified flows to recover within a reasonably short time period. The trade-off here is between the potential harm to legitimate flows by applying a too-restrictive throttling strategy and the lower protection provided to the victim by designing a fast recovery mechanism.

When the flow is classified as an attack flow for the first time after a long period, its rate is limited to a fraction of the offending sending rate. The size of the fraction is specified by the configuration parameter `DEC_SPEED`. Subsequent classification of a flow as an attack restricts the rate further, according to the formula:

$$rateLimit = \min(rateLimit, rate_{actual}) * DEC\_SPEED * \frac{B_{sent}}{B_{sent} + B_{drop}}$$

Thus the degree of the misbehavior of the flow defines the restrictiveness of the rate-limit. Flows that have worse behavior are quickly restricted to very low rates whereas this restriction is more gradual for better-behaving flows. Rate decrease is limited by the `MIN_RATE` configuration parameter so that at least some IP packets can reach the destination and trigger a recovery phase. Rate decrease progresses until the flow is classified as transient, at which point the slow recovery mechanism is triggered.

[3] This checking is necessary since the compliance of the flow parameters with the model can stem from the restrictive rate limit placed on the flow. If this is the case, misclassifying this flow as "normal" would lead to its removal from the cache at the end of the current observation period, and to removal of the imposed rate limit. This would quickly open a new opportunity for the attack.

The recovery phase is divided into *slow-recovery* and *fast-recovery* based on the values of the well-behaved parameter associated with the flow. During the slow-recovery phase, a flow is penalized for being classified as an attack flow by linear increase in the allowed rate according to the formula:

$$rateLimit = rateLimit + MIN\_RATE * \frac{B_{sent}}{B_{sent} + B_{drop}}$$

The duration of a slow-recovery phase is defined by two configuration parameters, `MIN_RECOVERY_PERIOD` and `MAX_RECOVERY_PERIOD`. The variable `period` is set randomly between these two values, to avoid synchronization of flows in the case of "pulsing attacks." After the flow has been classified as transient for `period` consecutive observation intervals (which is detected by the well-behaved counter having values higher than `period`), the fast-recovery phase is triggered.

During the fast-recovery phase the rate is increased exponentially according to the formula:

$$rateLimit = rateLimit * \left(1 + \frac{B_{sent}}{B_{sent} + B_{drop}}\right)$$

The rate increase is limited by the `MAX_RATE` configuration parameter. As soon as the rate limit becomes greater than `MAX_RATE`, the recovery phase is finished, the flow is transferred from transient to normal class, and the rate limit is removed.

There is a trade-off between low values of `MIN_RECOVERY_PERIOD`, which facilitate fast recovery of misclassified flows, and vulnerability to "pulsing attacks". Higher values of `MAX_RECOVERY_PERIOD` reduce this vulnerability by allowing more values for the `period` variable. They also keep the flow in the cache and enforce the rate limit. There is also a trade-off between high values of `MAX_RECOVERY_PERIOD` and the frequency of cache overflows, which reduce system performance.

The rate-limiting of the throttling component is very similar to the TCP's congestion control mechanism. It also performs an exponential decrease of the sending rate as response to detected communication problems, its slow-recovery period resembles the congestion-avoidance phase in the TCP during which the sending rate increases linearly, and the fast-recovery phase resembles the TCP's slow-start phase when the rate increases exponentially. A difference is that the rates regulated are the cumulative rates for the domain that correspond to many TCP and non-TCP flows. Another difference is that the speed of the rate change is guided by the flow's compliance to the imposed rate limit, thus the misbehaving flows endure a greater penalty.

The behavior of the system obviously depends on the proper parameter setting. The following section gives some guidance on the trade-offs involved, but the detailed evaluation of the effect of parameter values on the system functioning is part of a future work.

5. Experimental Results

Denial-of-service attacks occur in distributed and complex environments and can involve vast numbers of participating machines. Reproduction of similar conditions in a testbed environment is a challenging task. Testing systems for DDoS defense must include reproduction of real denial-of-service attacks. However, all the traffic produced during this testing must be limited to testbed machines to protect the rest of the Internet from congestion and avoid inflicting great damage if the experiment is badly designed. Few testbeds include more than several machines in a sufficient physical proximity to satisfy the previous requirement. Thus, reproduction of the scale of the attack is infeasible, and scalability and cost of any DDoS defense system cannot be well assessed through experiments.

Also, denial-of-service attacks disrupt legitimate traffic, either by denying the service to the victim or by creating congestion on intermediate routers. To assess the benefit of a source-end based DDoS defense system, we should measure the benefit or damage that such a system inflicts on legitimate traffic. This calls for generation or reproduction of legitimate traffic between the source network and the victim.

In our test runs the proposed DDoS defense system builds models of normal traffic from `tcpdump` traces collected at the exit router of UCLA CSD network. The models of normal TCP traffic are likely to be location

independent. The models of normal non-TCP traffic, on the other hand, must be generated from traces observed by the same source router that will later use them for flow classification.[4] Thus the source router in our testbed has to play the role of UCLA CSD exit router to be able to use these models. Also, during the test runs the source router must observe normal background traffic whose volume and dynamics correspond closely to the real traffic between UCLS CSD and the rest of the Internet. We handle these problems by profiling a different set of `tcpdump` traces than that used for model generation, and replaying them during the experiment.

Ideally, every line in a `tcpdump` trace should produce a packet that passes through the source router on its way to the destination and updates traffic statistics. However, the volume of traffic observed by UCLA CSD exit router is larger than testbed machines can efficiently handle; this replay process would quickly use up the source router's resources. To scale down this problem, we assume that the real router has a sufficient amount of resources to relay the traffic as efficiently during the attack as in normal operating conditions. Thus, the only flows that feel the impact of the attack and the source router's response are those flows that are either destined for the victim or generated by the victim. Only these flows need to be replayed through the network, since the attack and the response are likely to cause dropping of their packets, change of packet timing and change in the duration of connections. The rest of the `tcpdump` records can be read in directly from the trace and used to update the corresponding statistics.

The replay of flows associated with the victim starts with the choice of victim IP address and extraction of corresponding packet details from the trace. Two files are generated as the result of this process: `victim.trace`, which contains information for all packets sent by the victim to UCLA CSD hosts, and `ntg.trace` that contains all the information for UCLA CSD traffic to the victim. Two testbed machines are then chosen to play the role of the *victim* and the *normal_traffic_generator*. They are placed on different interfaces of the source router and replay the traffic from the generated files during the test runs.

Non-TCP packets can be described by timing, size, source and destination IP addresses and port numbers; and they are sent during replay according to the timing in the trace. TCP packets are further dependent on the state of the connection they belong to. To reproduce those connections correctly, we need to set up a TCP session between the victim and the normal_traffic_generator for every TCP connection in the trace. TCP connections are initiated at the time specified by trace, and data is delivered to the connection according to the timing of trace packets. TCP itself handles their exact sizes and the times they are injected into network, through the TCP's congestion window and acknowledgment mechanism. If the sending of some data packets is delayed, we shift the generation time of subsequent data packets from the same connection by this delay. In the test runs that do not involve attack packets, traces gathered during the replay process resemble, to a great extent, the real `tcpdump` traces used for their reproduction. During the attack, the TCP control mechanism responds to packets that are lost due to rate-limiting or congestion, by regulating the sending of new packets. We measure this impact and use this measurement as one of the indicators of the effect of our system on normal traffic.

Our testbed consists of a single source router, connected to eight testbed machines via Ethernet links. Out of those, one machine is designated to act as the normal_traffic_generator and the other seven are used for the attack. On another interface, the source router is connected to a machine acting as a victim. The source router is a Linux router, and it uses a Netlink interface to communicate with our DDoS defense system, which is implemented as a user application on the same machine. All testbed machines are identically configured with 1.4 GHZ AMD Thunderbird CPUs, 1GB RAM, and 100Mbs Ethernet NICs. To model the real network in which the router has greater resources than the victim, we artificially reduce the available bandwidth in the link from the source router to the victim. We generate the DDoS attacks by using real DDoS attack tools, such as TFN and Trinoo, and we intentionally test the attacks that invoke responses from the victim, such as TCP SYN and ping flood attack. We use 5 seconds as the observation interval, 3 as the maximum allowed packet ratio, 0.5 for the value of `DEC_SPEED` parameter, 60 for `MIN_RECOVERY_PERIOD`, 300 for `MAX_RECOVERY_PERIOD`, 20 for `MIN_RATE` and 100000 for `MAX_RATE`.

[4] Different "exit" routers are likely to observe different volumes of normal non-TCP traffic. For example, the level of normal UDP traffic is much higher for a domain containing large DNS-zone name-servers than for a college network.

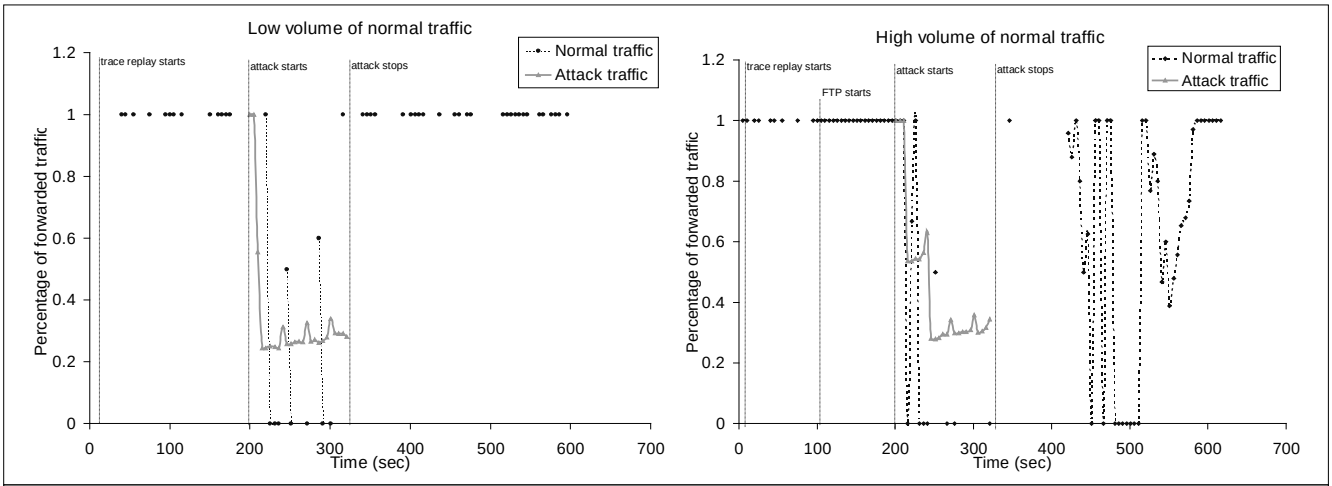

Figure 3: Effect of the defense system on legitimate and attack traffic

To measure the effectiveness of our system and the impact it has on attack and normal traffic, we ran several experiments in the identical setup, varying the level of normal and attack traffic. Figure 3 presents the result of two test runs. The defense system is started at the beginning of every run and given time to initialize. After 10 seconds, trace replaying is triggered on the victim and the normal_traffic_generator. The attack is started at 210 seconds and it lasts until 330 seconds. We continue measurement for 300 seconds after the end of the attack. During the first run, normal traffic to the victim was being replayed from `tcpdump` traces, and was on the order of several packets per second, containing no more than a thousand bytes each. The second run interleaved the same trace traffic and an FTP transfer of a 6.5MB file from normal_traffic_generator to the victim. Transfer is started 110 seconds after the beginning of the run. Figure 3 depicts the percentage of packets that were allowed to pass through the filter in every observation interval. In the case of low volume of background traffic, the attack is suppressed more efficiently and quickly (within three observation intervals) to about 30% of its capacity, where it remains until the end of the attack. At the same time, normal traffic suffers certain damage in some observation intervals but restores quickly after the attack has passed. In the case of high background traffic, the attack is suppressed more gradually, taking nine observation intervals to reach the 30% level, and normal traffic suffers greater damage in the recovery phase since its desired transfer rate does not comply with the restrictive rate limit. The difference in the speed of the response comes from the fact that high TCP background traffic in the second run generates large number of response packets from the victim, which facilitate the classification of the offending flow as compliant (210 to 230 seconds) and slow down rate-limiting. Figure 4 shows the change of rate limit over these two runs, and the desired and achieved rate values for the total flow to the victim (normal and attack traffic combined). In the case of high volume traffic, the sending rate violates the linearly increasing rate limit at 450 seconds, thus slowing the recovery of normal traffic. The rate

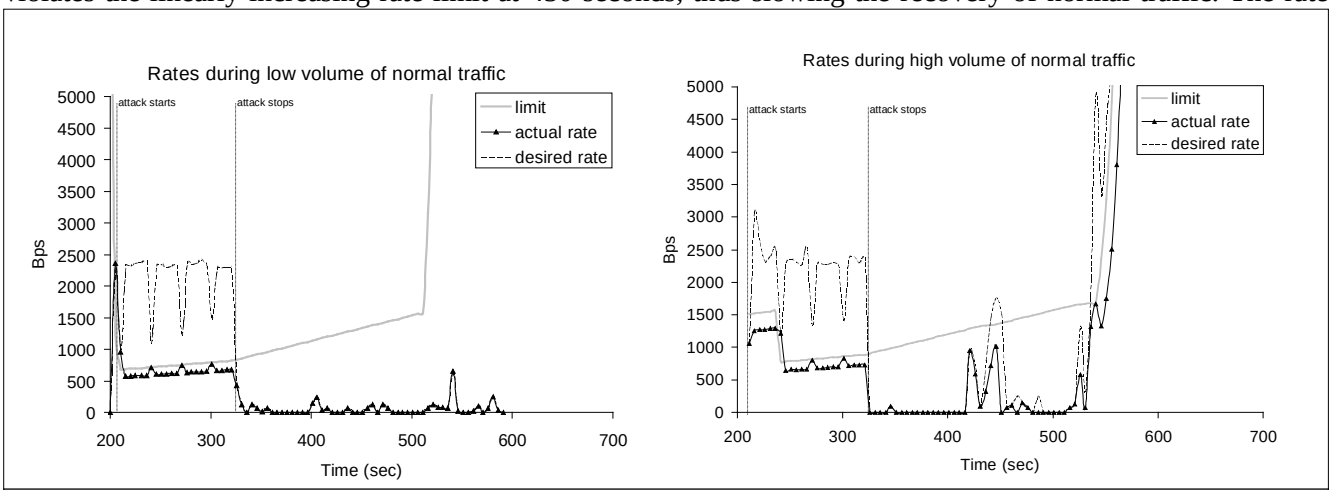

Figure 4: Rate limit, the desired and actual rate for the case of low and high volume of normal traffic

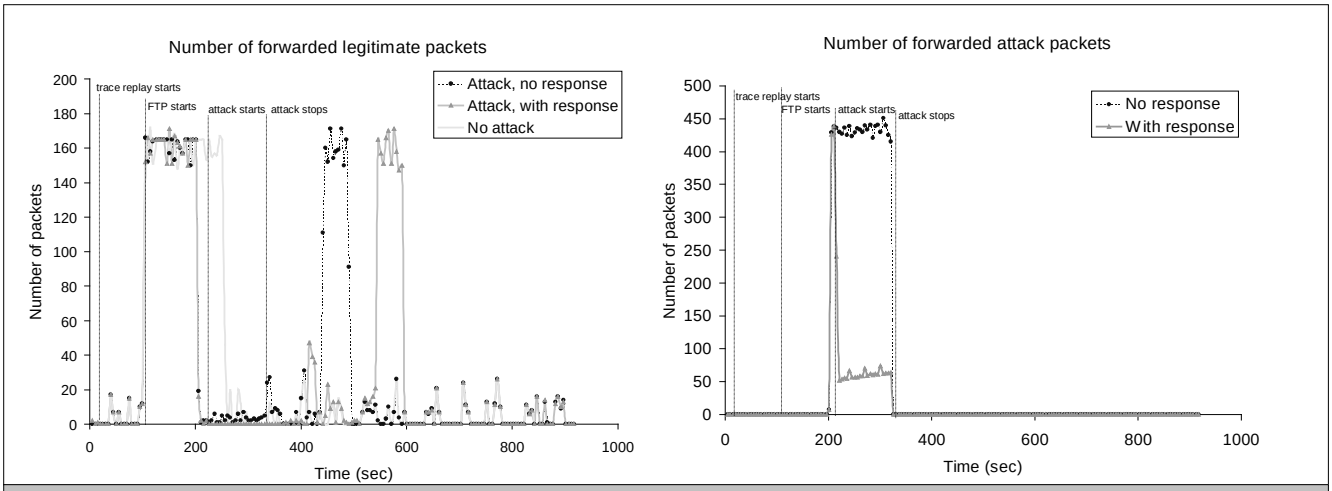

Figure 5: Number of forwarded packets in the case with and the case without response to the attack.

limit also reflects the misclassification of the attack as compliant flow in the second run, from 210 to 230 seconds, where it increases linearly.

We then evaluated the impact of the proposed DDoS defense system on the duration of legitimate TCP connections. We generated high volume of background traffic by interleaving the FTP transfer of the same 6.5 MB file with replayed traces, and observed the interaction of the normal with the attack flow and with the imposed rate limit. The number of normal and attack packets allowed to pass the filter in each observation interval for three test runs is shown in Figure 5. In all runs the FTP transfer is started at 110 seconds. In the first run there is no attack and no rate limit is imposed. In the second run the attack occurs from 210 to 330 seconds, and our system is not deployed on the router. In the third run, our system is deployed and does respond to the attack by rate-limiting the total flow to the victim. The FTP transfer in the first run ends at approximately 250 seconds. The onset of the attack in the second run takes up the bandwidth from the normal flow, and FTP reduces the sending rate in response to lost packets. After the end of the attack, normal communication is resumed and FTP increases the sending rate quickly and finishes transfer around 500 seconds. In the third run, the restrictive rate limit delays the recovery of FTP flow by additional 100 seconds, and the transfer finishes at 600 seconds. The delay stems from the aggressive increase in the sending rate by FTP, which does not comply with linear increase of rate limit in slow recovery phase. On the other hand, the benefit of rate-limiting the attack flow is evident from the second graph; it is quickly reduced to one tenth of its original sending rate and it remains there until the end of the attack.

We further evaluated the improvement of attack detection by using models of normal non-TCP traffic compared to detection based on disproportion in the numbers of sent and received packets. We targeted an attack on a

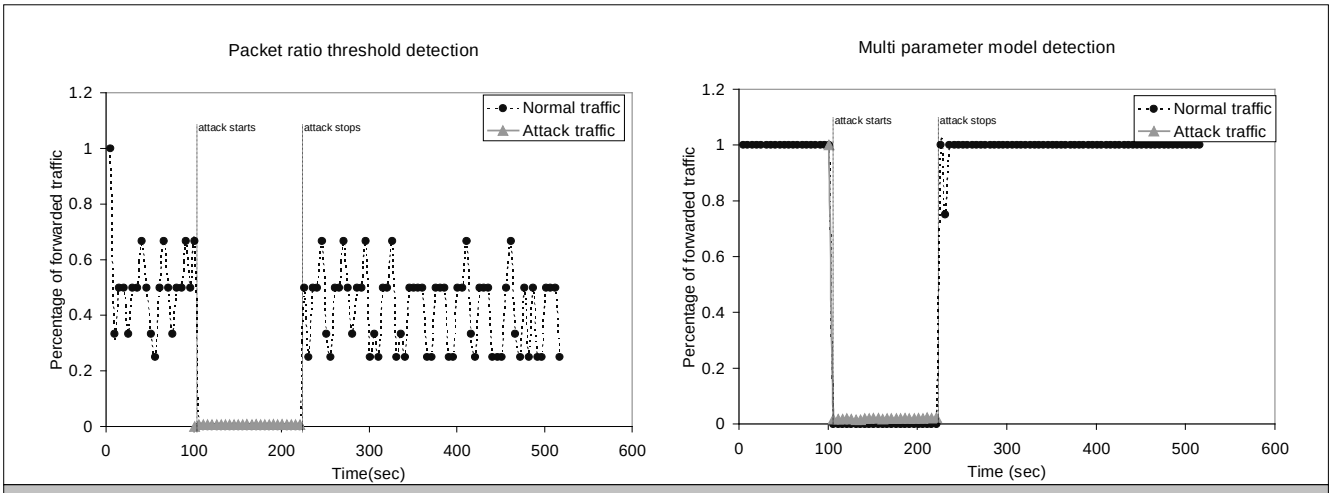

Figure 6: Effect of different detection mechanisms on normal and attack traffic

UDP destination and measured the levels of legitimate and attack traffic that were allowed to pass the filter during the attack and recovery phase. Trace replaying is triggered 10 seconds after the start of the run. The attack is started at 100 seconds and lasts until 210 seconds. We continue measurement for 300 seconds after the end of the attack. In the first run we use a disproportion in the number of sent and received packets to classify the flow as an attack flow, whereas in the second run we use precalculated models of normal non-TCP traffic for the given destination. Figure 6 presents the results of the experiment. In both runs, the throttling strategy is identical, and the attack traffic is quickly detected and dropped. When a fixed packet ratio threshold is used for detection, legitimate UDP packets are dropped constantly, since they do not produce enough reverse packets to comply with the allowed packet ratio. The drop rate is around 50% even when the attack is not going on. In the case where models of non-TCP traffic are used, legitimate traffic is dropped only during the attack and for a short time after the attack has finished, due to slow recovery phase.

The experimental results show that the mechanism outlined in this paper can effectively detect and handle individual DDoS flows at a router close to their source by comparing their two-way traffic parameters to models of normal activity. Even in the presence of high level of non-attack traffic, the system can throttle attack flows down within a few seconds, although the response is slowed down if non-attack traffic invokes a lot of replies from the victim. If applied to all flows in a DDoS attack, this form of throttling would make the attack much less effective. While the mechanism also throttles good flows for the same destination, it does not entirely shut them off.

The recovery of legitimate TCP connections after the attack can be slowed down by the system. The delay imposed on legitimate connections depends on the length of slow-recovery phase and the aggressiveness of connections. Less aggressive connections experience small or no delay since they do not try to send over the imposed rate limit, which facilitates faster increase of the rate limit. More aggressive connections violate the rate limit, which in turn slows down recovery process, and is amplified by the conservative reaction of TCP congestion control mechanism to lost packets. The behavior of the system can be tuned by changing the values of `MIN_RECOVERY_PERIOD` and `MAX_RECOVERY_PERIOD` configuration parameters. Lower values of these parameters facilitate fast recovery of legitimate TCP connections. However, they also endanger the victim's recovery during the attack, because the attack flows can recover quickly and deliver another full blow at the victim before their rate is limited again. The system is friendly to non-TCP traffic and does not misclassify it as an attack.

While the results are promising, they point the need for further research. More aggressive throttling strategies need to be deployed in cases of continued bad behavior, to avoid recurring attacks. Distinguishing between normal and attack flows within the aggregate flow to the victim would offer better fairness to normal flows. It would also facilitate lower delay on recovery of legitimate TCP connections after the attack, while still providing high protection to the victim. The separate strategy should be defined to handle non-TCP flows whose model does not exist in model cache, and the automated mechanism for model updating should be designed.

6. Related Work

While the intermediate routers that carry DDoS traffic might occasionally experience congestion due to large traffic volume, denial-of-service attacks inflict the greatest harm on the victim. Therefore, most of the systems for combating DDoS attacks have been designed to work on the victim side. As mentioned before, due to a large volume of the attack traffic, the DDoS defense system at the victim side cannot handle the attack. We discuss here the possibility of leveraging the research in this field at the source end.

Much of the work related to DDoS defense has been carried on in the area of intrusion detection. Intrusion detection systems detect DDoS attacks either by using the database of known signatures or by recognizing anomalies in system behavior. One might argue that it suffices to take a well-developed network-based intrusion detection system and deploy it on the source side, reversing its functions to examine outgoing traffic, to achieve the functionality similar to the system proposed in this paper. However, the attack appears different at the source and at the target network. At the target network all DDoS flows converge and affect the system greatly so that detection is inevitable; at the source end those flows are still dispersed and can appear as a set of perfectly valid transactions. It is usually the sheer amount of these transactions that saturates the target network.

Several popular network monitors perform mostly signature-based detection with some limited statistical processing such as CISCO's NetRanger [1], NID [11], SecureNet PRO [12], RealSecure [14], and NFR-NID [16]. Most of these systems do not take automated action to stop the attack, but just raise an alert to the system administrator.

In [10] an on-off control approach is proposed to fight DDoS attacks in a target network. In this approach, a router detects that it is the target of a DDoS attack by monitoring its buffer queue size. Once the queue size grows over a specified threshold, the router reacts by switching to a different mode of operation and throttling incoming traffic. Only packets belonging to certain type of traffic identified as problematic are dropped. When the queue size is reduced, throttling is stopped. This approach is similar to the RED congestion control mechanism [17]. Its application to the intrusion detection domain could efficiently protect the target of the attack from overflowing its buffers by early detection of DDoS attack, while at the same time not completely cutting off the traffic from the source network. However, its application at the source network of a DDoS attack would not be so effective, since the originating network does not experience sufficiently high traffic loads to trigger the response. It should also be noted that high traffic at a source network cannot be regarded as reliable sign of DDoS attack – it could be an excess amount of legal traffic that the destination network can easily handle.

Porras and Valdes [8] have recognized that the analysis of TCP/IP packet streams through the network could be beneficial to fighting intrusion attacks. This idea is similar to our approach and is implemented in EMERALD, an environment for anomaly and misuse detection ([5], [6], [7], [8]) developed at SRI. It combines a signature-based approach to IDS with statistical analysis for anomaly detection. EMERALD has many similarities with our system in that it performs statistical analysis of network traffic and is able to take an automated response when the system intrusions are detected. However, its statistical subsystem would have to be substantially modified to detect anomalies in two-way traffic.

Bradley et al. have proposed WATCHERS [3], an algorithm to detect misbehaving routers that launch denial-of-service attacks by absorbing, discarding or misrouting packets. WATCHERS uses the conservation of flow principle to examine flows between neighbors and endpoints. Hughes et al. have shown [4] that WATCHERS makes several implicit assumptions that do not hold and have proposed modifications to correct this thereby increasing the cost and complexity of the algorithm. Even with these modifications, WATCHERS requires explicit communication between routers. Algorithm cannot detect packets with forged source addresses. Even worse, such packets could be used by the attacker to misidentify a target as a bad router. WATCHERS can only detect compromised routers; it is helpless against misbehaving hosts. It also assumes that every router knows the topology of the network, which is not a feasible solution for large networks.

CISCO routers have built-in features such as debug logging and IP accounting that can be used for characterizing and tracing common attacks [2]. An access list can be configured to log many network events, e.g. to count packets received and categorize them by type, and to log sources of packets that are of special interest. This feature only gathers statistical data and offers no automated analysis or response. However, these features could be easily extended to implement our approach thus adding an analysis and response layer to existing routers.

In [18] Floyd et al. have proposed to augment routers with the ability to detect and control flows that create congestion and that are frequently part of DDoS attack. Flows are detected by monitoring the dropped packets in the router queue and identifying high-bandwidth aggregates that are responsible for the majority of drops. The rate limit is then imposed on the aggregate. If the congested router cannot control the aggregate itself, it can ask the adjacent upstream router to impose the rate limit on that aggregate. This upstream rate-limiting is called pushback and can be recursively propagated until it reaches the routers in the source networks. This approach offers a powerful tool to not only combat DDoS attacks but to also locate and remove network congestion caused by any other reason. However, it requires significant augmentation of the routers on the whole path from the victim to the sources. A single legacy router on the path complicates the scheme and imposes the need for secure communication of non-adjacent routers which makes the system vulnerable to attacks. The approach also requires cooperation of different administrative domains, which is currently hard to achieve.

In [19] Gil and Poletto propose a heuristic and a data-structure (MULTOPS) that network devices can use to detect DDoS attacks. Each network device maintains a multi-level tree that monitors certain traffic characteristics and stores data in nodes corresponding to subnet prefixes at different aggregation levels. The tree expands and contracts within a fixed memory budget. The attack is detected by abnormal values of packet ratio, and offending flows are rate-limited. The system is designed so that it can operate as either a source-end or victim-end DDoS defense system. While the high-level design of this system has a lot in common with our approach, the details are different. Non-TCP flows in system using MULTOPS can either be misclassified as the attack flows by the detection mechanism, or recognized as special and rate-limited to a fixed value. In the first approach, harm is done to a legitimate flow, while in the second approach a sufficiently distributed attack can still make use of the allowed rate to achieve the effect. The paper does not offer enough details on the rate-limiting mechanism, and does not address the removal of the rate limit after the attack, so we could not perform full comparison with our system.

Several systems combat DDoS attacks by using traceback mechanisms to locate attacking nodes ([29], [30]). These systems provide the information about the identity of attacking machines, but do not themselves stop DDoS attacks. The complexity of traceback mechanisms is large if the attack is distributed, and they may be susceptible to attempts by attackers to deceive them. Nonetheless, a good traceback system would complement our system very effectively.

Several filtering mechanisms have been proposed to prevent spoofing of source address in IP packets ([23],[24], [31]). While IP spoofing is not necessary for DDoS attacks, it helps the attackers to hide the identity of attacking machines so they could reuse them for future attacks. Eliminating the IP spoofing would complement nicely the proposed system. It would facilitate easy distinction of normal from attack flows, and reduce greatly the damage to normal flows during attack and recovery phase.

7. Security

Attackers have so far demonstrated the ability and willingness to develop more and more sophisticated attack tools that defeat current approaches to combating DDoS attacks. It is therefore expected that they will try to defeat the system proposed in this paper or to misuse it for denial-of-service attacks. Special attention must be paid to secure the system against such attempts.

The system operates autonomously and does not rely on communication with other entities. Thus its operation can only be affected by packet flows observed by the source router. An attacker that can control the reverse flow (from the rest of the Internet to the source network) could either prevent detection of the attacks by generating fake response packets, or even deny any service to clients from the source network by dropping the legitimate response packets to the network. Spoofing of packets is still possible at a large number of domains, and therefore the former threat seems very realistic. Wider deployment of mechanisms that prevent spoofing ([23], [24], [31]) can help reduce this problem.

The attackers could also perform “pulsing attacks.” Once the attack is suppressed by rate-limiting, they could abort it for a sufficient interval, and then restart it with full force. The victim would thus be periodically overwhelmed with attack packets for the short interval, until the DDoS system detects and limits their flows. The random choice of penalty period can help in breaking up the synchronization of attack flows from different source networks, thus possibly taking the full force out of the attack. However, more sophisticated techniques are needed to solve this problem completely.

The attackers could choose to merely use up a lot of the victim's resources, thus degrading the service instead of denying it completely. Our system would not be able to detect such attacks since there would not be any signs of difficulties visible at the source end. The extension of the system with a capability to act on the request of the victim, if sufficiently secured, could handle this problem.

Finally, the attackers might try to perform a DDoS attack on the source router. While the size of memory structures is limited and cannot be controlled by the attacker, it is possible that a vast number of packets from the source network would exhaust router’s processing resources. However, the gain from this attack is small. By disabling the source router, attackers lose their connectivity to the world and cannot perform any more attacks.

8. Implementation Considerations

If the system proposed here were deployed in a small number of source networks, the DDoS threat would not be significantly reduced. Since only a few source networks would throttle attack flows, attackers would merely find networks where the system was not deployed. Thus, a high degree of deployment is a condition for the high effectiveness.

Deploying a DDoS defense system on core routers would have the significant advantage of policing a large fraction of the total Internet traffic; thus a large coverage could be achieved with a small number of deployment points. However, core routers must operate at high speeds and have very few resources to spare for tasks other than routing. The memory and processing overhead that a defense system introduces per packet would have to be quite small in order to avoid performance degradation. On the other hand, faster processing and smaller state lead to poor detection of the attacks, and misclassification of flows at the core router would damage a significant portion of the legitimate Internet traffic. Deploying a defense system at the source router achieves smaller coverage for the same number of deployment points, but avoids performance degradation and can host a more complex detection system for higher accuracy.

To encourage source networks to deploy the DDoS defense system proposed in this paper, we attempt to introduce minimal changes to the source router and offload the processing and storage tasks to the separate units, as shown in Figure 1. The following are necessary functions that the source router should provide: (1) passing of the whole packets or packet headers to the observation component on a separate link and (2) implementation of per-destination rate limits. The source router should further be able to communicate with the throttling component to obtain new rate limits and implement them in its filtering mechanism. It is desirable for the source router to also provide some feedback as to which packets were dropped due to rate limits, to facilitate monitoring of the functioning of the defense system. To the best of our knowledge, today's routers already possess these functionalities, or can be easily extended to provide them.

The additional incentive to deploy the defense system might be the low cost and easy installation of its components, small or no changes to the existing routers, good performance and low impact on normal functioning of the source network. However, these factors merely make deployment less painful. The source networks do not protect their own nodes from DDoS attacks by deploying this or similar systems, and thus currently have low motivation to do so. However, it can be expected in the future that legal or governmental factors will lead network operators to install systems that make their networks harder to use as attack points. The Internet has become such a vital part of business that denial-of-service attacks and network outages cost companies immense amounts of money, and something will be done to ensure that those connecting to networks behave well. In such an environment the implementation of the proposed DDoS defense system would be highly beneficial to the source networks as well as to their peers.

9. Future Work

Denial-of-service attacks are complex and resistant to many security mechanisms. The proposed system does not offer a complete solution to them, but rather presents a new approach to the combat. While the performance results are promising, there are a number of open problems.

The detection of the attack is not completely reliable, and misclassification of normal flows is still possible. The extension of a statistical model would improve the accuracy of classification, but would increase storage and processing demand on the source router. Testing of the system on a real router would help assess the realistic storage and processing overhead and find the optimal setting that achieves high accuracy at an acceptable cost. We are working on the implementation of our system on an Intel IXA IXP programmable router to evaluate this trade-off.

The system tries to combat “pulsing attacks” by randomizing the length of the slow-recovery period. While this might help to break the synchronization of the attack flows from different source networks, it still does not guarantee sufficient security to the victim. A better mechanism is needed to recognize repeated attacks and react more restrictively to those.

A serious drawback of the proposed system is that it can only detect the attack after it has seriously affected victim's resources. While most attacks achieve their full power quickly and aim at taking up as much resources as possible, "smarter" attacks could aim at the slow degradation of the victim's services. These attacks might be hard to detect by the proposed system, and require design of more sophisticated detection mechanisms.

During the attack, the imposed rate limit is applied to the aggregate flow toward the victim of the attack. The legitimate flows compete for the available bandwidth in the source router with the aggressive attack flows and usually receive much less than their fair share. Ideally, the DDoS defense system should be able to recognize legitimate flows and let them through while only policing the attack flows. We are working on strategies that would help us differentiate legitimate from attack flows and guarantee better fairness of our response. The traceback of the attack is also a desired feature of the source-end based defense system.

Finally, automatic update of normal traffic models to reflect changing trends in normal traffic would make the system evolvable. The special care must be taken to prevent *retraining* of the models by the attackers.

10. Conclusion

We have designed and implemented a prototype of a DDoS defense system based on the concepts of detecting misbehaving traffic close to the source by observing two-way statistical behavior, and throttling data flows that are suspect. We have tested the prototype and demonstrated that it can be effective in handling real DDoS attack tools in a testbed environment.

This approach is a useful adjunct to existing DDoS defense tools, such as ingress filtering and traceback systems. It requires more development before it is ready for use in the field, but the preliminary experiments reported here suggest that the approach can indeed be effective. Routers deploying such a system could be confident that their networks were not being used effectively to launch DDoS attacks. A secondary benefit would be that their inefficacy for attacking would make their nodes less desirable targets for compromise.

The system is practical in its use of resources and other requirements. For its intended deployment targets, statistics can be updated quickly enough and throttling is possible. Hardware trends in routers provide the tools needed to support this system.

Substantial research remains to complete the system, including development of more sophisticated models and throttling capabilities, closer examination of security issues, and work to differentiate attack flows from other flows destined for an attack target. The results presented here suggest that the final system will be extremely effective at slowing down DDoS attacks to manageable levels. When such attacks are no longer greatly effective, they are likely to disappear entirely.

11. References

[1] Cisco, "NetRanger Overview", http://www.cisco.com/univercd/cc/td/doc/product/iaabu/csids/csids1/csidsug/overview.htm

[2] Cisco, "Characterizing and Tracing Packet Floods Using Cisco Routers", http://www.cisco.com/warp/public/707/22.html#2b

[3] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson, "Detecting Disruptive Routers: A Distributed Network Monitoring Approach," Proceedings of the 1998 IEEE Symposium on Security and Privacy, May 1998.

[4] J.R. Hughes, T. Aura, and M. Bishop, "Using Conservation of Flow as a Security Mechanism in Network Protocols." Proceedings of the 2000 IEEE Symposium on Security and Privacy, May 2000.

[5] P.G. Neumann and P.A. Porras, "Experience with EMERALD to DATE", 1st USENIX Workshop on Intrusion Detection and Network Monitoring, April 1999.

[6] U. Lindqvist and P.A. Porras, "Detecting computer and network misuse through the Production-Based Expert System Toolset (P-BEST)", Proceedings of the 1999 IEEE Symposium on Security and Privacy, May 1999.

[7] P.A. Porras and P.G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", Proceedings of the Nineteenth National Computer Security Conference, October 1997.

[8] P.A. Porras and A. Valdes, "Live traffic analysis of TCP/IP gateways", Proceedings of the Symposium on Network and Distributed System Security, March 1998.

[9] S.Dietrich, N. Long and D. Dittrich, "An Analysis of the "Shaft" distributed denial of service tool", http://www.adelphi.edu/~spock/shaft_analysis.txt

[10] S. Liu, Y. Xiong, and P. Sun, "On Prevention of the Denial of Service Attacks: A Control Theoretical Approach", IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, June 2000.
[11] Computer Incident Advisory Capability, "Network Intrusion Detector Overview", http://ciac.llnl.gov/cstc/nid/intro.html.
[12] MimeStar.com, "SecureNet PRO Feature List", http://www.mimestar.com/products/
[13] CERT Coordination Center, " CERT Advisory CA-2001-19 'Code Red' Worm Exploiting Buffer Overflow In IIS Indexing Service DLL", http://www.cert.org/advisories/CA-2001-19.html
[14] Internet Security Systems, "Intrusion Detection Security Products", http://www.iss.net/securing_e-business/security_products/intrusion_detection/index.php
[15] CERT Coordination Center, " CERT Advisory CA-1999-17 Denial-Of-Service Tools", http://www.cert.org/advisories/CA-1999-17.html
[16] NFR Security, "NFR Network Intrusion Detection", http://www.nfr.com/products/NID/
[17] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” IEEE/ACM Transactions on Networking, August 1993.
[18] S.Floyd et al, “Pushback Messages for Controlling aggregates in the Network", Internet draft, Work in progress, http://search.ietf.org/internet-drafts/draft-floyd-pushback-messages-00.txt, July 2001.
[19] T. M. Gil and M. Poleto, "MULTOPS: a data-structure for bandwidth attack detection", Proceedings of 10th Usenix Security Symposium, August 2001.
[20] Internet Traffic Archive, "LBL-PKT Traces", http://ita.ee.lbl.gov/html/contrib/LBL-PKT.html
[21] Internet Traffic Archive, "DEC-PKT traces", http://ita.ee.lbl.gov/html/contrib/DEC-PKT.html
[22] National Laboratory for Applied Network Research, "NLANR Packet Header Traces", http://pma.nlanr.net/Traces/Traces/long/auck/2/
[23] J. Li, J. Mirkovic, M. Wang, P. Reiher and L. Zhang, "SAVE: Source Address Validity Enforcement Protocol", Proceedings of INFOCOM 2002, June 2002.
[24] P. Ferguson and D.Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", RFC 2827.
[25] D. Dittrich, "The DoS Project's 'trinoo' distributed denial of service attack tool", http://staff.washington.edu/dittrich/misc/trinoo.analysis
[26] D. Dittrich, "The 'Tribe Flood Network' distributed denial of service attack tool", http://staff.washington.edu/dittrich/misc/tfn.analysis.txt
[27] D. Dittrich, "The 'Stacheldraht' distributed denial of service attack tool", http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt
[28] D. Dittrich, "The 'mstream' distributed denial of service attack tool", http://staff.washington.edu/dittrich/misc/mstream.analysis.txt
[29] S. Bellovin, M. Leech and T. Taylor, “ICMP Traceback Messages”, Internet draft, Work in progress, http://search.ietf.org/internet-drafts/draft-ietf-itrace-01.txt, October 2001.
[30] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. “Practical Network Support for IP Traceback,” Proceedings of ACM SIGCOMM 2000, August 2000.
[31] K. Park and H. Lee. “On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets,” Proceedings of ACM SIGCOMM 2001, August 2001.