Intrusion Detection Routers: Design, Implementation and Evaluation Using an Experimental Testbed

Eric Y. K. Chan, H. W. Chan, K. M. Chan, P. S. Chan, Samuel T. Chanson, M. H. Cheung, C. F. Chong, K. P. Chow, Albert K. T. Hui, Lucas C. K. Hui, S. K. Ip, C. K. Lam, W. C. Lau, K. H. Pun, Y. F. Tsang, W. W. Tsang, C. W. Tso, D. Y. Yeung, S. M. Yiu, K. Y. Yu, and Weihua Ju

***Abstract*—In this paper, we present the design, the implementation details, and the evaluation results of an intrusion detection and defense system for distributed denial-of-service (DDoS) attack. The evaluation is conducted using an experimental testbed. The system, known as intrusion detection router (IDR), is deployed on network routers to perform online detection on any DDoS attack event, and then react with defense mechanisms to mitigate the attack. The testbed is built up by a cluster of sufficient number of Linux machines to mimic a portion of the Internet. Using the testbed, we conduct real experiments to evaluate the IDR system and demonstrate that IDR is effective in protecting the network from various DDoS attacks.**

***Index Terms*—Distributed denial-of-service (DDoS), intrusion detection, routers, testbed.**

I. Introduction

A. Background

In a distributed denial-of-service (DDoS) attack, a group of well-organized and widely distributed zombies simultaneously and continuously send a large-volume flood of packets to a victim host or network. This overwhelms the victim who cannot serve its legitimate clients. Usually, by the time the attack is detected, there is not much a network administrator can do except manually disconnect the victim from the Internet and seek help from their upstream Internet service providers (ISPs).

Starting from February 2000 when the famous e-commerce sites shutdown events were reported, DDoS attacks continue to disturb the existing Internet infrastructure [32], [41]. With the increasing numbers of system vulnerabilities that ease network intrusion and more sophisticated DDoS attack tools publicly available to script kiddies, the damaging effect of the attack is not only increasing in monetary losses [12] but governments are also aware of the threat of bringing down the critical information infrastructure that can affect national security and loss of life [13].

Manuscript received September 15, 2005; revised March 30, 2006. This work was supported in part by the Areas of Excellence Scheme established under the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-01/99), and in part by a grant from the Innovation and Technology Commission of the HKSAR, China, under Project ITS/170/01.

E. Y. K. Chan, H. W. Chan, K. M. Chan, P. S. Chan, M. H. Cheung, C. F. Chong, K. P. Chow, L. C. K. Hui, S. K. Ip, C. K. Lam, K. H. Pun, Y. F. Tsang, W. W. Tsang, C. W. Tso, S. M. Yiu, K. Y. Yu, and W. Ju are with the Department of Computer Science, The University of Hong Kong, Hong Kong (e-mail: smyiu@cs.hku.hk).

S. T. Chanson, A. K. T. Hui, and D. Y. Yeung are with the Department of Computer Science, The Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong.

W. C. Lau is with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong.

Digital Object Identifier 10.1109/JSAC.2006.877214

B. Related Work

Different from the classical denial-of-service attack, DDoS attack packets are characterized by its huge volume and the absence of packet content signature, except the possibly forged source for hiding zombies' real locations. Therefore, it is not effective to apply rule-based pattern matching system to counteract the attack. Moreover, applying ingress filtering [16] to prevent packets with forged source Internet protocol (IP) addresses to be sent out cannot stop nonspoofed DDoS attack traffic. Some researchers proposed to defense the DDoS attack from the end-host perspective [6] that is only specific for some attacks targeting on a particular network protocol.

Since routers are part of critical network components that forward packets within the Internet, various researchers proposed to tackle the problem at router level in recent years. The techniques used in these approaches include the followings. Packet-marking that overwrites the content of existing packet header or modifies existing Internet protocols for defending [42]. Tracing sources of attack, [5], [34], [39], [40] tries to identify the sources of attack in order to stop it. Pushback router [29] issues rate-limit request to its upstream routers for traffic control upon detecting a DDoS attack. It provides an infrastructure that is suitable to operate within an organization. However, pushback request may not easily reach upstream router through the congested link. This also punishes legitimate traffic that shares the same path with attack traffic. DWARD [31] is a typical example of a source-end detection router which is usually deployed at the outgoing gateway of a subnet and will try to detect outgoing attack packets. Other references include [19], [23], [35], and [43]. There are many other works in this area and we only mentioned the ones which are strongly related to our work.

II. Problem Definition

DDoS attack is a complex problem due to its distributed nature and the attack packets involved usually do not have content-specific signatures. It is not easy to derive an effective solution. We first divide the problem into several subproblems.

A. Identify Attack and Victim

The first challenge in solving the DDoS attack problem is to determine if there is any DDoS victim being attacked from the traffic passing through a router. This problem introduces several technical difficulties. It is not trivial how to locate a victim from a sea of network traffic flow through a router within the Internet. It is important to have this information available efficiently, accurately, and in a scalable manner. In a backbone router that

connects to Internet, the possible number of destination IPv4 addresses is in the order of magnitude of 2^{32} and router links can allow high-speed traffic with rates up to 10Gb/s for OC192 link. It is difficult to keep track of this amount of traffic statistics per pair of source and destination addresses within any network components. When a victim is identified, we need to determine how much of traffic volume flow to the target victim should be classified as suspicious.

B. Classify Legitimate and Attack Traffic

Once a potential victim is discovered, it is important to carry out defense mechanisms to allow the victim host or network to be accessible for legitimate use. This presents a problem of how to differentiate attack traffic from legitimate traffic efficiently and accurately in order to reduce collateral damage. It is particularly important in the case that the victim is a popular site that would face flash crowd traffic. As pointed out in [11], intruder continues to combine different multiple types of packet streams as attack option and also there exist attack tools that can self-propagate and produce new types of attacks. Therefore, it becomes a key issue for the detection mechanism to classify new type of attack traffic. This also requires a notion of what legitimate traffic is.

C. Defense Attack

The DDoS defense mechanisms should minimize the damage of the attack traffic to the end point victim, as well as intermediate network infrastructure (e.g., routers). Automatic and real-time response is necessary; or it will be too late for the network administrator to rescue the victim. Due to the distributed nature of the attack, the defense mechanism should be easily deployed in both edge network and backbone network without incurring high cost and influencing the existing Internet infrastructure in the aspects of, for examples, router loading and existing network protocols. The research presented in this paper focuses on the detection and defense mechanism. We do not address the traceback problem that locates sources of zombies and hackers.

III. The Methodologies Used

A. Attack and Victim Identification

In order to identify the existence of DDoS attack and the target victim, we combine the space efficient Bloom filter [7] data structure and leaky buckets algorithm [2] to monitor bandwidth usage of network traffic passing through the router that targets to a particular destination. The usage of Bloom filter has been growing in networking community in recent years (e.g., [14] and [15]) to detect nonresponsive traffic and heavy flows, respectively. One property of our Bloom filter is that we measure arrival rate of incoming traffic based on packet destination address or prefix and take into consideration that traffic can be burst due to various network behavior such as transmission control protocol (TCP) slow start, retransmission, or bursty application [22].

Bloom Filter: We implement Bloom filter as an array of L groups of N buckets (see Fig. 1) to summarize how much traffic going to a particular destination within a fixed period. This allows more effective access of the array in parallel although it has a slightly higher probability than the usual one-dimensional "power-two-based" Bloom filter on false positive [8]. In each bucket, we keep the following values: a *packet counter* to record how many packets that have arrived within the period, an *theoretical arrival time* to determine if a packet has arrived too early to make the bucket overflow, and a *threshold* value to decide how serious of bucket overflow should be classified as unexceptional heavy volume of traffic flow.

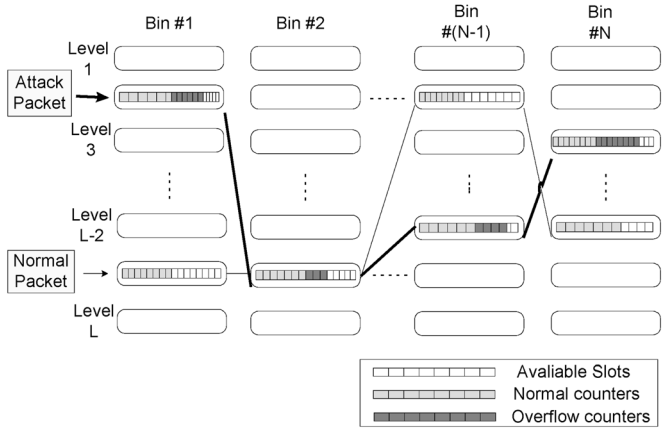


Fig. 1. Bloom filter.

Leaky Bucket: Initially, all values are set to zero. When incoming packets are delivered to the router, L independent hash values are computed from their destination address prefixes to identify the target victim network (to identify victim host, it can simply set the address prefix length to be 32). Each hash value is used as index to one of N buckets in each group for incrementing stored packet counter. Moreover, the theoretical arrival time value is incremented with a fixed amount of service time if the packet arrives to the mapped bucket later than the existing stored time value or slightly earlier than the theoretical arrival time. However, if a packet arrives too early before the theoretical arrival time and wait over a fixed period of time, the packet is classified as an overflow packet and the stored *overflow counter* is incremented. If overflow counters exceed the specified threshold values in each mapped L buckets, it is considered to have an unexceptional heavy volume of traffic.

Hashing Scheme: To achieve better performance and the ease of future extension to exchange Bloom filter detection information among routers, we avoid executing L independent hash functions on packet but repeatedly iterate a simple shift-register random number generator [30] L times to generate L random values based on packet's destination IP address prefix and a predefined constant as a seed. With this scheme, we can quickly compute L hash values and can easily modify the behavior of the generator by changing the predefined seed regularly to avoid hacker to guess the behavior of the detection mechanism.

Adaptive Threshold Scheme: Since network traffic volume varies from time to time and has bursts, it is impossible to use a fixed threshold value to identify heavy traffic flow because simply setting a very small value will generate many false positive results. Similarly, setting a very large value will generate many negative results. We used an adaptive approach by periodically perform sampling to make better estimate of threshold values in each bucket to reflect the current situation. We borrow the idea of TCP mean round-trip time and round-trip timeout

estimation [21] to determine a dynamic overflow threshold as follows. We keep an *expected overflow count* E in each bucket, and it is set to zero initially. In each sampling period, the exponential weighted moving average of the value is updated according to the formula $E_{\text{new}} = w \times M + (1-w) \times E_{\text{old}}$, where M is the *overflow counter* stated in previous paragraph, and the value w indicates the importance between current measurement and the estimated value. We also compute mean deviation D in each interval as $D_{\text{new}} = D_{\text{old}} + u \times (|M - E| - D_{\text{old}})$, where u also weights the current measurement against the previous estimation.

For simplicity, we assume both w and u share the same value. Based on these values, we can set up the threshold value $T = E_{\text{new}} + n \times D_{\text{new}}$ adaptively, where n denotes the number of allowed deviations. If the overflow counter exceeds the current T value, traffic currently mapped to the bucket is classified as unexceptional heavy if it exceeds the expected number of arrival packets plus some amount of mean deviations. For sites that have low volume of traffic flow, this scheme may lead to a small threshold value, which cause false alarms easily when normal traffic flow through a site increased. Therefore, some preconfigured lower bound is defined to automatically upgrade small threshold value. Similarly, a preconfigured upper bound threshold value is necessary to avoid too many false negatives. Currently, sampling works in discrete interval to avoid too frequent reset on the Bloom filter. This causes small number of attack packets passing through when the volume of attack do not attain to certain threshold.

B. Legitimate and Attack Traffic Classification

Once a DDoS attack and target victim are identified, we try to identify attack traffic and legitimate traffic from the detected heavy volume traffic streams so further actions can be taken to protect both legitimate traffic and network infrastructure. Although DDoS attack packets look legitimate, its massive volume significantly affects the usual distribution of various packet attributes within a period. Therefore, we build up the notion of legitimate traffic at routers under usual situation and perform fine-grain analysis on suspicious traffic to identify anomalies from the built model.

Baseline Traffic Model: It is observed that distributions of some invariant packet attribute values in legitimate traffic are usually clustered and quite stable from site to site [24], [28] under normal situation. One typical example is that without Smurf attack or user datagram protocol (UDP) flood attack, the number of TCP packets should dominant against UDP and Internet control message protocol (ICMP) packets. Similarly, the ratio of SYN flag against ACK flag and others in TCP packets would increase drastically under SYN flooding attack. Therefore, we characterize the legitimate traffic model by measuring the distribution of different packet attribute values flow through the router. We call it the baseline traffic profile.

Then, anomaly can be identified from legitimate traffic by measuring the deviation. Since single attribute cannot reflect the whole traffic characteristics and attacker can combine multiple types of packet streams for attack, we construct the baseline profile by combining distributions of multiple attributes as in [26]. With this scheme, it is difficult for attacker to determine all such site-specific information or generate attack stream to the site that match all distributions.

TABLE I
TRAFFIC CHARACTERISTICS MEASURED FOR DDOS PACKET CLASSIFICATION

CHARACTERISTICS MEASURED IN PROFILE	RATIONALE FOR SELECTION
Percentage of different IP protocols values versus total number of IP packets	To detect substantial change of packets with unusual protocol values, e.g. UDP flood.
Percentage of different IP payload length values versus total number of IP packets	To detect substantial change of unusual size of packets.
Percentage of different time-to-live values versus total number of IP packets	To detect how far the packets are coming from and what kind of IP stack generate these packets.
Percentage of different type-of-service values versus total number of IP packets	To detect substantial change of type-of-service value in IP packets.
Percentage of different TCP source ports values versus total number of TCP packets	To detect substantial change of TCP based application usage.
Percentage of different TCP destination ports values versus total number of TCP packets	To detect substantial change of TCP based application usage.
Percentage of different TCP flags values versus total number of TCP packets	To detect abnormal change of TCP based connection behavior.
Percentage of different UDP source ports values versus total number of UDP packets	To detect substantial change of UDP based application usage.
Percentage of different UDP destination ports values versus total number of UDP packets	To detect substantial change of UDP based application usage.

Table I shows the list of traffic characteristics we measured in baseline profile. Both the distribution of source and destination IP prefixes are also good representatives of network traffic flowing through router. However, it requires a lot of memory to represent the distribution of them and it is still under investigation. Therefore, we do not include IP prefixes in our current measurement.

Due to the large number of packet attribute values in each characteristic but only some of them are dominant (usually known as iceberg values), we currently expect network administrator to partition packet attribute values into groups for profiling. To solve this problem in the future, we plan to use schemes [3], [25], [24] that determine those frequently appeared iceberg entries automatically and measure their statistics. In order to train the legitimate traffic model and minimize the day-of-week and time-of-day effects, we adopt sliding window approach to collect traffic statistics. Traffic in the Internet would have expected fluctuation of arrival rates among a period of time-of-day and day-of-week, so applying sliding window technique would avoid using a fixed absolute value for all periods and allow the acceptable variations from one period to the other. The scheme divides each fixed time period as a fixed number of equal-sized time-based windows. When a packet arrives to a router, measured packet attributes values are extracted and corresponding counters are updated accordingly. When profiling process is completed, the moving average and deviation of each (group of) attribute value(s) are collected.

Our baseline profile is designed to allow traffic measurements from all the egress ports or a specific port. The advantage of using a single profile to approximate traffic characteristics

flowing through all ports reduce the memory requirement of an intrusion detection router (IDR) to keep track of possibly large number of histograms in per port profile, while using per port profile, specifically collected for a network port, represents traffic characteristics more accurately.

Packet Scoring: Beside the baseline traffic profile that is built offline, we also collect runtime traffic statistics from the suspicious packet streams detected from the Bloom filter mechanism. Due to limited processing capacity in router, this runtime profile is built with shorter measurement period, and in discrete-time interval, instead of sliding window based as compared with the baseline traffic profile. The baseline traffic profile can be viewed as the expected likelihood of each packet attribute values in legitimate traffic, while the runtime profile measures the observed occurrences in current traffic. Therefore, we define an anomaly score for each packet attribute value as $|m-\mu|/\sigma$, where μ, σ represent the expected mean value and standard deviation of packet attributes values (ratio) in baseline profile, respectively, while m represents the observed mean occurrence. This formula heuristically quantifies how much packet attributes value deviate from its normal profile. Moreover, an overall anomaly score is assigned for each suspicious packet by computing the weighted sum of anomaly scores gained based on the packet's attribute values. The larger score value a packet has, the more deviate from the expected values specified from the normal baseline profile, and more likely to be abnormal. Thus, it is classified as attack packet if its score value is higher than a threshold value defined by the site administrator. Currently, we use a static threshold value. In the future, we will consider adjusting the threshold value dynamically.

Robustness: It may be argued that the scheme can be defeated if a sophisticated attacker knows the approximations of baseline profile. However, note that the distribution of packets attributes such as time-to-live (TTL), and those ratios are quite site-specific. They are difficult for outsider to collect. By deploying multiple IDRs to form security perimeters within an organization, it is quite hard for attackers to use zombies distributed over the Internet to prepare traffic that satisfy these multiple attributes-based profiles kept in all detectors without being detected. There may also exist slow attackers that attempt to change the baseline profile so that they appear legitimate. However, it may take a long time for the attackers to do so, and the system should have a threshold to bound the baseline profile so that whenever the threshold is reached, the administrator should be aware of the situation.

C. Mitigation of DDoS Attack

Once attack traffic is classified, defense actions should be automatically taken place to mitigate the impact of the attack. Fig. 2 shows the standalone defense mechanisms implemented in the IDR system. By discarding those attack packets, this not only prevents the bad packets from reaching target victim but also dramatically reduces the workload of routers that forward them and avoid wasting the network bandwidth in downstream network. Hence, this can effectively protect the overall network infrastructure.

We apply the well-known class-based queueing (CBQ) technique [17] to perform bandwidth management on the link that forwards those low scored suspicious packets, including those

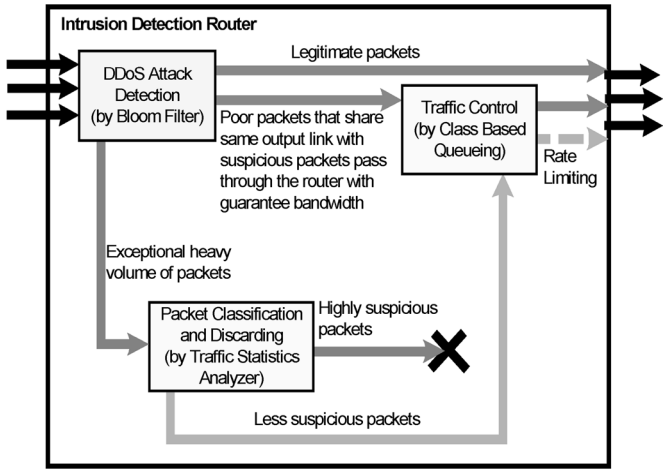


Fig. 2. Defense mechanisms in IDR.

legitimate packets sharing the same destination with the attack packets or poor packets that unfortunately map to all congested buckets in the Bloom filter. The packets are regarded as poor if the following occurs. When the Bloom filter parameter is set with the destination address prefixes identify the target victim network instead of the target victim. Then, both legitimate and attack packets towards victim network would share the same output link. The legitimate packets are considered as poor and are rate-limited. In any case, this scheme ensures guaranteed bandwidth can be given to poor packets that share the same output link with suspicious packets, and thus reduce collateral damage on legitimate traffic. Suspicious packets are put to a separate queue for rate limiting. The amount of bandwidth to be given would be the affordable limit at the victim side so that legitimate traffic can reach the victim, while unclassifiable attack traffic cannot bring down the victim.

The automatic packet dropping and rate-limiting defenses are effective to mitigate the impact of DDoS attack and reduce the workload of router. With the autonomous nature of IDR, it is easy to deploy many IDRs in the upstream autonomous system (AS) networks to form distributed security perimeter and defeat the DDoS attack at the earlier stage.

To summarize, the use of Bloom filters, leaky bucket, hashing scheme for attack, and victim identification allow the system to realize an attack with very little overhead in memory and computation. For the classification of legitimate and attack traffic, the baseline traffic model provides an adaptive scheme to fit different networks, while the technique of packet scoring can effectively classify a normal packet from an attack packet. Finally, the use of CBQ approach can mitigate the impact of DDoS attack, while reducing the workload of router. For these components to work smoothly, we need to integrate them together nicely into the system. We will highlight and discuss how to implement these components to obtain an effective system in Section IV.

D. Deployment Discussion

We believe that DDoS attack problem should be solved in a distributed way. We propose to deploy IDRs from the border routers of each AS down to its internal routers to form levels of security perimeter for protecting from DDoS attack.

At the AS level, each IDR protects their downstream network by monitoring traffic flow based on destination IP prefix instead of a full 32-bit destination IP address. This keeps the memory requirement of the Bloom filter at a low level for detecting attacks on multiple victims, while maintaining a low false positive rate. Analytically, the probability of classifying legitimate packet as attack packet on the same set of buckets is $P = (1-(1-1/N)^M)^L$, where N is the number of buckets in each level, L is the number of levels in the Bloom filter, and M represents the number of flows being classified as heavy flows. It can see that even we use a Bloom filter of 4 levels and 1024 buckets, which occupies only 144 kB memory in our implementation but with less than 1% false positive rate for protecting 256 potential targets or IP prefixes within the AS. At the downstream routers that close to the stub network, IDRs can choose per destination IP address as heavy traffic flow measurement. This could quickly identify any victim within its network.

We can also measure and keep baseline traffic profiles based on the hierarchy or different length of network prefix within the AS infrastructure. Routers near the AS border keep only aggregated baseline profile for the network, while routers near the stub network keep profiles per host or standard "template" profile that describe maximum allowed traffic towards end host inside the stub network. These profiles can be stored at separate control server to offload the memory requirement of IDRs. IDR can query these control servers for required profile through private channel when suspicious DDoS attack occurs. As mentioned in the previous section, the profile size can be reduced using iceberg-style that maintains only distributions of frequent happen traffic characteristics values. We believe that this effectively reduce the memory requirement of a router to store many different type of baseline profiles. The remaining problem is that when a router is already suffered from DDoS attack and has not yet query control server for baseline profile that is used to discard packet. This can be solved by always keeping one per router-based profile at the router that measure traffic flow through the IDR. The IDR can use this local baseline for initial packet discarding to reduce its loading during DDoS attack, until it can spare CPU power to query control server for destination-based profile. Then, a more fine-grained packet classification can be performed.

To form a security perimeter within the AS, we can apply scheme similar to the pushback design [29]. That is, downstream IDR propagates attack and defense related information to all upstream routers once DDoS attack is identified, in case the attack volume is not sufficiently concentrated for detection at upstream. This information includes victim destination prefix, scoring information, and scoring thresholds from downstream router. Upstream IDRs can calculate scores for each incoming packet with matched destination IP prefix, and then discard or perform rate limit on those attack packets. However, all communications should be done in private channel because the link between downstream routers can be already congested due to the attack before attack can be mitigated successfully. In addition, peer AS border IDRs can exchange these attack and defense information so that attack can be suppressed as close to the attack sources as possible. Downstream router can periodically query status of upstream routers regarding the volume of dropped packets in order to determine the DDoS attack has been stopped or not. This effectively coordinates routers to perform an online, DDoS defense.

We consider the design of communication protocols and the distributed detection and defense cases as our future work. Therefore, the experiments we describe in Section V simply apply standalone IDRs in the network topology to show the effectiveness of mitigating DDoS attack.

IV. Implementation Issues

In the following, we highlight the issues related to implementation of IDR.

1) Detecting DDoS should be done in real-time. If the implementation is done in user-level, the efficiency is a bit slow. So, we propose to implement the core DDoS attack detection and defense in kernel mode.
2) The implementation of Bloom filter adaptive leaky bucket should allow frequent access without a lot of page faults. So, we carefully control the size of the Bloom filter table and allocate expensive contiguous memory for it inside the kernel.
3) For the traffic control, based on an existing Linux package, iproute2, we port some of the programs in the package to kernel module so that we can invoke the CBQ mechanism within the kernel while allowing operator to manipulate the CBQ parameters.

Kernel Versus User Space Implementation: In order to test the effectiveness of the IDR system for defeating real-time DDoS attack, we implement the schemes on Linux router equipped with dual CPUs as prototype. Our schemes are initially implemented as user-level program based on libpcap library [27] instead of using network simulator approach [33] because we want to measure system's performance such as CPU loading and network throughput in handling realistic attack besides the detection accuracy. The benefits of developing the system and all core schemes in user-level provide us a stable environment for debugging and testing with both deterministic traffic traces files and realistic network. However, it introduces critical performance problem in real-time detection and defense due to large overhead of copying packets from kernel to user-level during DDoS attack. Also, we cannot find a way to perform packet discarding at user-level based on Linux kernel 2.2. In fact, there are other problems with this version of kernel.

We finally chose the Linux kernel version 2.4.18-17, which was the latest one at the time of development. The Linux 2.4 kernel provides a netfilter [37] framework that allows kernel modules to perform further mangling packets beside the normal network stack operations without modification on the original kernel source tree. This version also reimplemented the bottom halves as "multithreaded" softirqs so the network stack can run simultaneously on all CPUs.

The current IDR system consists of four major components: Linux kernel module that performs core DDoS attack detection and defense, Java-based network management client that presents various status of monitored routers to remote network operators graphically, a daemon-based agent that runs on each router and perform polling on the kernel module and sent detection results to client regularly, and a user-level traffic profiling tool that collects baseline profile. We concentrate the kernel module details in the following paragraphs.

The kernel module is designed modularly to allow incorporation of any new algorithms easily. It consists of three parts: the intrusion detection module (IDM) that implements the Bloom filter adaptive leaky bucket (BFALB) algorithm for detecting DDoS attack and victim, the traffic statistic analyzer (TSA) that performs fine-grained traffic analysis and discarding suspicious packets, and the output traffic control (OTC) module performs CBQ scheduling on suspicious packets. To make the kernel module upgradeable for newer kernel version, we set up "/proc" file system for both system configurations and runtime status reporting instead of introducing any new system call. Also, individual parts are able to turn on or off for changing various parameters and running standalone for experimental evaluations.

Various kernel data structures are created dynamically to support runtime configurations, including Bloom filter that contains an array of adaptive leaky bucket cells, event queue that aggregates numerous number of similar DDoS attack alerts based on victim IP address and packet protocol value before reporting, and traffic profiles that represents baseline profile and real-time profile.

Bloom Filter Leaky Bucket: We use a Bloom filter table of 4 levels and 1024 bins that the requests of dynamic contiguous memory allocation can be satisfied inside kernel while maintaining low false positive rate. For kernel that allows up to 512 contiguous memory pages to be requested, we limit the maximum dimension of Bloom filter table up to a total of 8 levels and 4096 bins which requires only 2 MB memory. We believe that this is sufficient to produce a very small false alarm on multiple attack streams. We allocate expensive contiguous memory for the Bloom filter within kernel because it is the most frequently accessed structure. We do not want possibly frequent page faults.

A fixed size circular event queue is employed to aggregate huge amounts of similar DDoS alert information, and thus avoid frequent reporting as a way to deny the service of the IDR. Currently, the textual alert information is reported at "/proc" file system and is regularly requested from IDR agent daemon for notification to GUI client. Destination IP addresses, IP protocol value, and timestamp of attack packets is inserted into the queue in case the previous attack event does not share the same set of information. The packet timestamp denotes either the first or latest detection time in the event structure and the number of this event occurrence is also increased. When there is no more empty slot in the event queue for new entry, the existing entries except the latest ones are removed and this involves updating the queue's head counter and length only. Currently, the system is configured to aggregate 10 000 similar attack packets as an event and the queue keeps 100 entries.

To reinitialize the leaky bucket cells for sampling, a kernel timer is triggered at a fixed interval. Currently, the interval is set at 5 s. Setting it to a small value may lead to too many software interrupts and incur large overhead for detection such as wasting CPU cycles to perform concurrency control. Setting it to a large value may accumulate more legitimate packets that are hashed to same buckets and cause bucket overflowed or congested.

Each traffic characteristic within baseline and real-time traffic profile is implemented as fixed length array of buckets because packet attributes distribution is dominated by some values and some less frequently appeared attribute values can be shared with the same bucket. We assume that network operators can collect profile to deduce the maximum number of icebergs. The maximum array size currently is set to 128 so that it can handle sufficient number of icebergs values. In order to update bucket value, we current scan on the array to locate the bucket within the array due to the possibility of range settings in some buckets.

Since the main purpose of real-time traffic profile is to prepare anomaly score for packet attribute values, we use the packet arrival time to estimate the start and end of window frame for the profiling period instead of generating kernel timer interrupt. We believe that it is simple enough and avoid generating too many software interrupts.

Usage of Netfilter: To examine all sanity checked packets coming from the network, we implement the BFALB algorithm at the netfilter's NF_IP_PRE_ROUTING hook and set a mark to the memory buffer (sk_buff) of each suspicious packet. This mark is useful at a later stage for traffic control. Moreover, marked packets are passed to the Traffic Statistics Analyzer for analysis and scoring within this hook. The kernel module either discards high scored packet by returning the NF_DROP value inside the hook or lets low scored packets to continue traversal within kernel by returning the NF_ACCEPT value. It offloads the CPUs from further process (such as forward those attack packets inside the kernel) and mitigate the problem of receiving network interrupts continuously (known as livelock problem) in a general way instead of those hardware specific solution described in [30].

Currently, we observed that layer four's packet header information is not extracted to the memory sk_buff buffer due to the IP layer nature of the netfilter system. Therefore, we expect that there should be fewer layer four packets that would fail from sanity checks and rejected through the scoring mechanism.

We placed another hook at the NF_IP_FORWARD point to create queueing disciplines to rate limit low score suspicious packets. Issues on using the queueing disciplines within kernel will be further described in this paper.

Traffic Control Implementation: We apply the Linux iproute2 package [1] that provides implementation of various packet scheduling methods to offer major traffic control features such as queueing disciplines, classes (within a queueing discipline), filters, and policing. The package can be divided into kernel part that interacts with network device to schedule outgoing packets, and user-level part that validates operator's traffic control commands and converts parameters to fit with kernel processing. In our application of CBQ to control link bandwidth sharing between suspicious packets and legitimate packets, we set up a root queueing discipline with maximum allowed bandwidth of link device. The purpose is to control overall transfer rate for outgoing device that forwards both suspicious and legitimate packets. A child class is constructed to limit the suspicious packets with transfer rate set by network operators. This rate is a value safe to the downstream victim and would not cause significant collateral damage on legitimate traffic to the victim, such as 50% of link bandwidth. Based on the mark placed from the BFALB algorithm as handle at an earlier stage, we can set up a filter to efficiently distinguish packets and put to different queueing disciplines.

One possible implementation of applying iproute2 package is to run a user space daemon that wait for kernel request on setting up CBQ commands and parameters upon DDoS attack. This requires either new definition of real-time kernel message based on mechanisms like rtnetlink socket and system call, or let the daemon poll system "/proc" file regularly and invoke the commands. The first way is not portable, while the second one seems not effective if everything is already working inside kernel but defense mechanisms parameters are passed between kernel and user space for derivation. We notice that the user space tc program performs a lot of floating point computation for deriving appropriate values for CBQ parameters such as maximum idle time [18] that should be avoided in kernel development [36], and the intelligent queueing disciplines black box is only callable through rtnetlink mechanism from user-level. Instead, we choose another approach by porting part of the tc program as part of our kernel module so that we can invoke the CBQ mechanism within the kernel while still allowing the network operator to manipulate CBQ parameters. The porting mainly involves four parts.

1) Perform careful integer approximation on some CBQ parameters by scaling and rewriting the formula. The error due to our integer approximation way on deriving CBQ parameters is expected to be 1%–2% for some cases.
2) Mimic the nonexported part of the rtnetlink mechanisms so that the queueing disciplines can still be invoked through rtnetlink mechanism in kernel.
3) Implement a kernel thread that invokes queueing disciplines upon detection of attack. This is because directly calling on queueing disciplines library within the bottom halves is prohibited by the existences of some nonatomic memory allocations found in queueing disciplines routines in call path.
4) Reduce the usage of limited kernel stack memory by modifying some of tc library that declare large local array within routines that are in function call path at runtime.

Note that the system uses a number of counters to keep track the traffic. If the counters are not reset from time to time, it may have an overflow problem. To tackle this issue, we can do the following. Assuming that we need to keep track of the traffic for at least x consecutive hours, we divide x into, say t periods. For each period, we keep separate counters to store the values. We accumulate the values in the original counter as time goes by. Whenever overflow occurs (can be easily detected), we delete the values for the earliest k periods. The number of bits of the counter, the values for x, t, k should be tuned according to the real situation.

V. Experiments and Results

In order to test the behavior of IDR systems in detecting and defending realistic DDoS attacks, we built a Linux testbed that has 21 software routers and 65 end-host desktops to simulate a portion of Internet. Software routers and end hosts are connected together through CISCO Catalyst 2980G-A and CISCO 2950-T network switches with virtual LAN technology enabled. The CISCO 4006 router is connected to outside network. Fig. 3 shows the experimental platform for the experiments. The software routers are DELL PowerEdge1650 rack mountable servers equipped with dual Pentium-III 1.4 GHz CPU, 512 MB PC-133 SDRAM, two Intel Pro100S dual port FastEthernet adaptor, and two 1Gbit/s fast Ethernet port. The end hosts are composed of both Desktop PC equipped with Intel Pentium-4 1.7 GHz CPU, 256 MB memory, integrated 3Com 10/100 Ethernet card and AMD Duron 800 MHz CPU, 128 MB memory, and 100 Mbits/s Realtek Ethernet card. The machines are installed with RedHat 7.3 and Linux kernel 2.4.18.

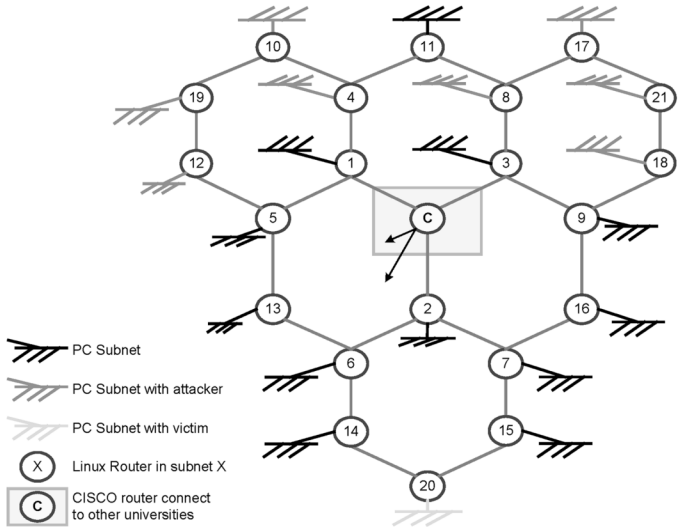


Fig. 3. Linux cluster for experimentation.

Internet application traffic including web, e-mail, ftp, news, and DNS and UDP streams are generated among end hosts as legitimate background traffic, except subnet 11 that currently connects the testbed to the Intranet and provides master services like DNS, NIS, and NTP to the whole testbed. Moreover, both backbone and software routers are operating under OSPF protocols. For the traffic sent from the top to the bottom within the topology, routing tables are configured to pass through middle software routers such as 5 and 9 instead of the CISCO router currently so that we can have better insight of the Linux router behavior under DDoS attack. The background traffic in each subnet is generated continuously and randomly with a global set of parameters such as number of servers and their location for connections, numbers of files allowed for download, number of e-mail sizes, etc., such that the average loading of about 10 Mbits/s traffic is generated from each subnet. This traffic generation will introduce a small degree of variations in the packets generated, which should not impose significant impact in our measurements.

We distributed zombies evenly at the top of the topology and attacked victim at the bottom. Our experiments apply the TFN2K [4] attacker tool to generate TCP SYN flood attack. There will be 8 zombies located at subnet 12, 19, 10, 4, 8, 17, 21, and 18. The attack is commanded from subnet 11 and last for about a 15 min interval. The attacker sends at its full speed to deliver about 9000 packets/s. Note that due to the limitations of the routers, the communication links and the switches, the current testbed cannot handle traffic of 10 Gb/s which is the real traffic volume in the Internet. However, our testbed should be able to provide better insights to the investigator on the effectiveness of the proposed schemes than a pure simulation. Currently, attack packet size is of 1500 bytes. Results on slower attack rates and

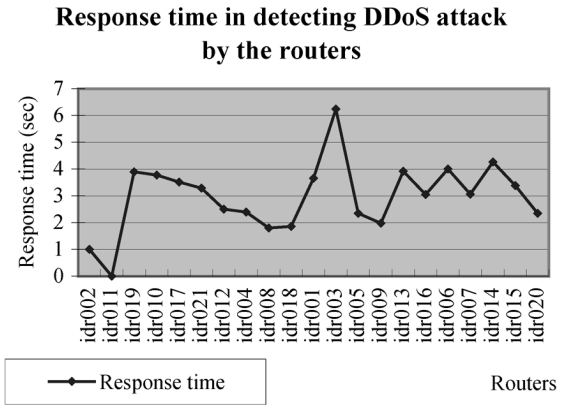


Fig. 4. Response time in detecting DDoS using BFALB algorithm.

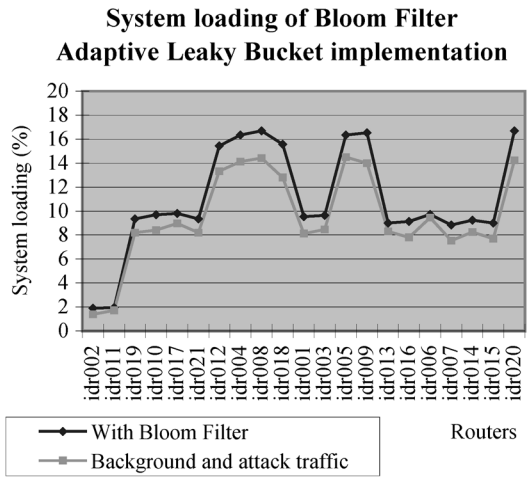


Fig. 5. System loading of the BFALB algorithm in detecting DDoS attack.

TABLE II
BFALB PARAMETERS USED FOR TESTING DDOS ATTACK DETECTION

Netmask length to detect portion of victim IP prefix	32
Number of levels in Bloom Filter	4
Number of bins in Bloom Filter	1024
Sampling period	5 seconds
Processing time used in leaky bucket	5000 μsec
Max waiting time used in leaky bucket	1000 μsec
Initial overflow threshold (within sampling period)	25,000 packets
Minimum overflow threshold (within sampling period)	25,000 packets
Maximum overflow threshold (within sampling period)	30,000 packets
New sampling weight	1
Deviation allowed	1

on different types of attacks will be briefly discussed at the end of this section.

Bandwidth and Victim Detection Experiment: Our first experiment shows that exceptional heavy volume traffic from DDoS attack the victim and can be identified in the presence of some legitimate background traffic by using the BFALB algorithm. In Fig. 4, we measure the time that the IDR system first detected the exceptional heavy flow. It is found that every router can detect the attack within 8 s. The x axis of the graph presents the router location in the topology where the expected amount of attack traffic is received in increasing order. We observe that there is an increase of response time for some downstream routers due to packet loss and congestion of some middle routers that have aggregated large amounts of attack packets.

Bloom Filter CPU Loading Experiment: To demonstrate that the BFALB detection algorithm can be applied to router with limited spare CPU cycle, we measure the incurred system loading of the IDR routers under DDoS attack using the set of parameters given in Table II. Fig. 5 reveals the system loading for each router with and without the IDR system under DDoS attack. From the graph, we can see that when a router received significant amounts of aggregated traffic, the system loading is increased significantly. However, overhead imposed by the Bloom filter algorithm is less than 2% and is not significant as compared with CPU consumption on handling traffic passing through it. The problem of packet loss also causes a drop of CPU loading in some downstream routers such as idr013 to idr015.

TABLE III
BASELINE PROFILING SETTINGS

Packet attributes	Values partition	Scoring weight
IP protocol #	0, 1, 2-5, 6, 7-16, 17, 18-127, 128-255	1
IP payload length	0-15, 16-31, 32, 33, 48-63, 64-127, 128-255, 256-511, 512-1023, 1024-1479, 1480, 1481-65535	1
IP TTL values	0-31, 32-56, 57, 58, 59, 60, 61, 62, 63, 64, 65-127, 128-191, 192-255	1
IP TOS values	0, 1-7, 8-15, 16-31, 32-63	1
TCP flags values	0-1, 2-3, 4-7, 8-15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25-31, 32-63	1
TCP source port values	0-19, 20-21, 22, 23, 24, 25, 26-118, 119, 120-255, 256-511, 512-1023, 1024-1999, 2000-2999, 3000-3999, 4000-4999, 5000-5999, 6000-6999, 7000-7999, 8000-8999, 9000-9999, 10000-19999, 20000-29999, 30000-39999, 40000-49999, 50000-59999, 60000-65535	1
TCP destination port values	Same as above	1
UDP source port values	0-52, 53, 54-1023, 1024-1999, 2000-2999, 3000-3999, 4000-4999, 5000-5999, 10000-29999, 30000-49999, 50000-65535	1
UDP destination port values	0-52, 53, 54-1023, 1024-1999, 2000-2999, 3000-3999, 4000-4999, 5000, 5001, 5002-9999, 10000-29999, 30000-49999, 50000-65535	1

Packet Classification Accuracy Experiment: In order to determine the accuracy of packet classification on suspicious packets, we perform baseline profiling on each router for 3 hours with window frame size of 600 s and step of 120 s to collect legitimate traffic model. The real-time traffic profile is built with window frame size of 60 s. Table III shows the profiling settings.

Then, victim is attacked by different kinds of attack streams and spoofed packets and we counted the number of misclassified packets based on our knowledge of our network IP prefix within a period of 30 min. We use the static threshold value 40 which is equivalent to the maximum score of any four attributes, respectively. Table IV shows the results at each router. IDR succeeds to differentiate legitimate and attack traffic under appropriate profiling and settings in most routers.

TABLE IV
PACKET CLASSIFICATION ACCURACY UNDER TCP SYN FLOODING WITH FULL ATTACK RATE

Router	Total suspicious packets passed to TSA	Total normal packets scored	Total attack packets scored	packets scored	No. of false negative packets scored	TSA false positive rate	TSA false negative rate
idr002	536053	47	88796	0	0	0.00%	0.00%
idr011	337523	9	21925	0	0	0.00%	0.00%
idr019	7453785	149	1397407	0	0	0.00%	0.00%
idr010	7288506	1409	1363748	0	0	0.00%	0.00%
idr017	7640832	317	1435428	0	0	0.00%	0.00%
idr021	7218589	115	1350274	0	0	0.00%	0.00%
idr012	12893224	230	2404395	0	0	0.00%	0.00%
idr004	12935878	1537	2412145	0	0	0.00%	0.00%
idr008	13368080	619	2499646	0	0	0.00%	0.00%
idr018	12492752	2616	2322919	0	0	0.00%	0.00%
idr001	7084480	1025	1320401	0	0	0.00%	0.00%
idr003	7388289	642	1381546	0	0	0.00%	0.00%
idr005	13616841	1419	2531616	0	0	0.00%	0.00%
idr009	13544664	2334	2515132	0	0	0.00%	0.00%
idr013	6709660	766	1243992	0	50257	0.00%	4.04%
idr016	6675969	1528	1236860	0	49845	0.00%	4.03%
idr006	7367485	1115	1377113	0	0	0.00%	0.00%
idr007	6561828	1316	1214624	0	0	0.00%	0.00%
idr014	6847608	1269	1272796	0	110606	0.00%	8.69%
idr015	6557558	1488	1213678	0	86657	0.00%	7.14%
idr020	13792055	2833	2563610	0	280203	0.00%	10.93%

Packet Classification Loading Experiment: We measure the extra CPU loading imposed by the packet classification mechanism on suspicious packet streams. Since IDR turns on packet classification mechanism only when it detects exceptional heavy traffic, discarding attack packets leads to smaller number of attack packets sent to the bottom of the topology and cannot reveal the full picture of down stream routers under heavy congestion. Therefore, our loading measurements are carried out without packet discarding. Fig. 6 shows the increase of system loading and actual system loading on each router. When more traffic are aggregated, the CPU consumption relative to total CPU time spent on forwarding packets increases. However, under current environment and setting, the extra loading of the packet classification mechanism is bounded by not more than 4%. When we apply the packet discarding defense mechanism, there is a significant drop in CPU loading compared to the case without defense because CPU times were saved from not forwarding those heavy volume of discarded attack packets.

Sustained Throughput With Complete Defense Experiment: To measure the sustained throughput after applying IDR defense mechanisms in all routers, we perform an ftp session in a subnet 1 (10.1.1.3) to download a file of size 17.2 MBytes from the ftp server inside an innocent server within victim subnet 20. The transfer rate is recorded. Fig. 7 shows the results. If there are attacks without defense, none of the legitimate FTP requests are connected successfully. When we switch on the IDR system for

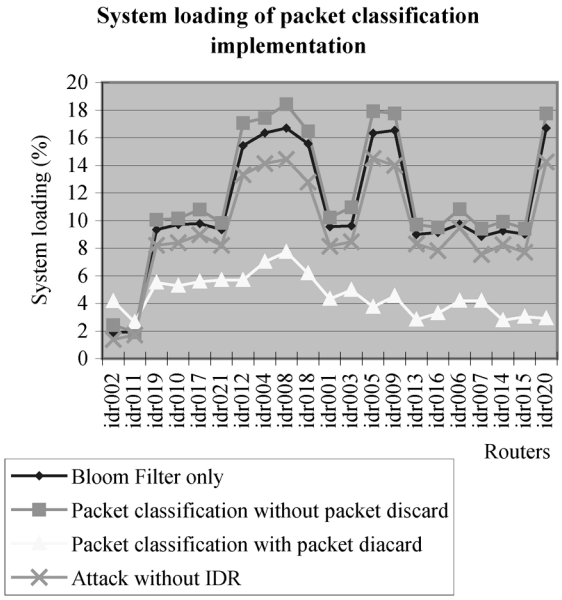


Fig. 6. Loading of packet classification with and without packet discard.

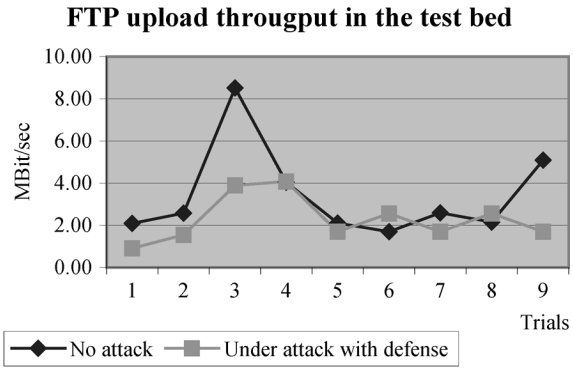


Fig. 7. FTP throughput on DDoS attacked testbed with defense enabled.

TABLE V
TRANSFER DELAY FOR DWARD AND IDR

	Average Transfer Time (sec)
No attack (for reference)	220.5
DWARD	838.2
IDR	547.7

defense, the average throughput of the system can retain about 1/2 of that of the system when there is no attack.

We also investigate the delay experienced by the system (see Table V). The delay is about double. All legitimate FTP requests are completed even under attack.

Sustained CPU Loading With Complete Defense Experiment: We also measure the loading of routers after applying all defense mechanisms in the IDR routers. Fig. 8 shows that with complete defense, the CPU loading of all the routers with DDoS attack packets passing through drops from 10% to 2% compared with the case without defense. This is because CPU times are saved from not forwarding those heavy volumes of discarded attack packets. No attack packets passed through router idr002 and idr011. Therefore, with complete defense, the CPU loading

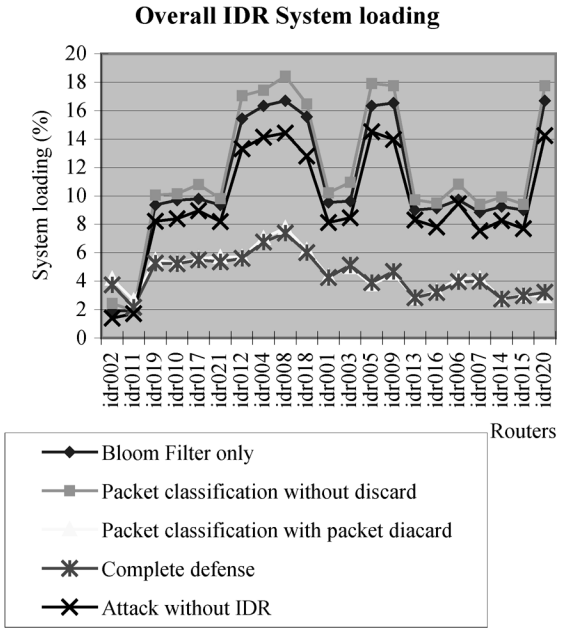


Fig. 8. Loading of routers after applying all IDR defense mechanisms.

raised because of the extra workload from the DDoS detection mechanism.

Attacks With Different Rates and Different Types: We have also performed experiments with attacks different rates and different types of attacks. We have tried to generate packets at the rates of 3000, 5000, and 9000 packets/s. For attacks, we have tried the generic attack (the attributes are set randomly), the SYN flood attack, the UDP flood attack, and a mix of all the above. The performance of our system is more or less the same for different types of attacks.

For attack rates, we observed that for slow attack rates, some of the IDRs (usually at upstream) may not regard the attack packets as harmful, so basically let the packets pass through. In fact, none of the normal packets has been dropped in these IDRS. In other words, the volume of the attack packets is not large enough to affect the normal operation of the systems. We also noted that some of the IDRs may accept attack packets but maintaining TSA false positive around 0%. In other words, almost all the normal packets are not dropped. This implies that as long as the attack traffic is not heavy enough, it will not affect the normal traffic a lot. In that case, accepting some attack packets would not be a problem. On the other hand, when the attacks are generated with full speed, the performance of the system is similar to that shown in Table IV.

Remark: We have also evaluated the performance of the system by setting different weights for different attributes under the four different attacks (UDP, TCP, ICMP, and the combinations of these three). Our preliminary results show that TTL is a critical attribute for differentiating the attack packets from the normal packets. On the other hand, we found that the system still works well by setting the weights to different values. We have tried to set the weight of TTL to vary from 1 to 4 and we still got a similar performance under the attacks In most of the cases, the threshold can remain as 40. In other words, the performance of the system is quite robust. Note that

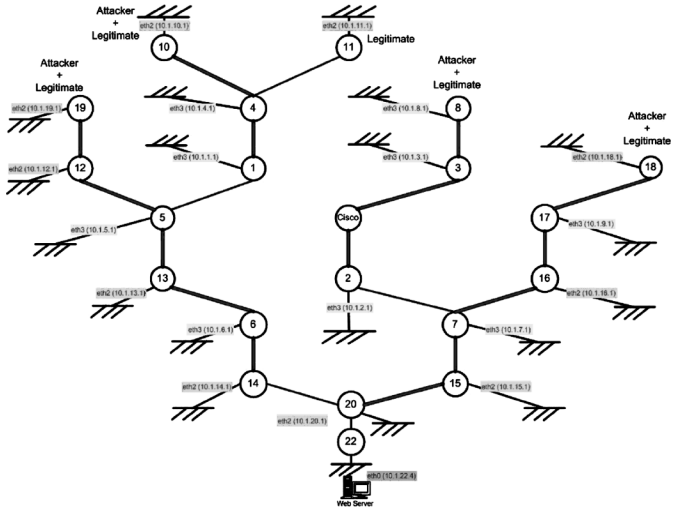


Fig. 9. Topology for comparing DWARD and IDR.

if TTL is spooled, it may affect the performance of the system. However, the system is based on the combination of attributes for classifying the attack packets. It may not be easy for the attackers to spool a packet that can escape the detection. On the other hand, we need to investigate further the effect of the attributes in order to have a better scheme to combine different attributes for the detection of attack packets.

VI. Comparison with DWARD

We have compared the performance of our system with DWARD. We have used the topology shown in Fig. 9 for the experiment. We have set up four DWARD routers deployed at the outgoing gateways of four different subnets (8, 18, 19, 20). When using IDR, we deployed three IDRs at the junctions where the subnets join (3, 5, 20). This setup should optimize the performance of both DWARD and IDR. Eight attackers are distributed in four subnets (8, 10, 18, 19). Each of these subnets has two attackers and one legitimate host. All attacks are towards the victim at Network 22. The attacks are generated at its full speed (about 9000 packets/s). The attack type is generic attack with random packet attributes (with TCP, UDP, ICMP). For DWARD, we use the default settings.

We then start ten ftp transfer sessions from a machine in Network 11 to get a 1.85 GBytes file from the victim. This legitimate ftp traffic will share the attack traffic on the same path towards the victim. The link capacity for the victim is 100 Mb/s, while the total attack traffic is about 56 Mb/s. We measure the following for comparisons, the CPU loading, the memory consumption, and the average delay in the ftp transfer. We have repeated the experiments ten times. Based on our experiments, we found that the CPU loading is at least five times more using DWARD than IDR. For the memory consumption, DWARD also uses more memory (several times more than IDR). Table V shows the average delay for the ftp transfer in the experiments. There are delays using either DWARD or IDR. The delay for IDR is a little bit shorter than DWARD. A similar result has been obtained by putting DWARD and IDR some other routers.

Note that this is only a preliminary comparison between the two systems. A more in-depth evaluation on these systems together with other intrusion detection systems should be carried

out in the future. Also, we realize that DWARD and IDR can be used together as their design principles are complementary. DWARD is a source-end detection router and it can do an excellent task to avoid attack packets to be sent out from its subnet. On the other hand, IDR is best to be deployed in ISPs so that when the attack packets from different subnets arrive at the junction, IDR can easily detect the attack.

VII. Conclusion

In this paper, we have discussed the design, the implementation, and the evaluation of an IDR based on a testbed that is made up of a cluster of linux machines. We show by real experiments, instead of simulation, that IDR is effective in protecting a network from DDoS attacks. Although the testbed is still far from the real Internet, by performing real experiments in the testbed can help investigators to have a better understanding on the actual performance of the proposed schemes. We believe that based on the experience of this testbed, a better one can be designed and constructed to represent a more realistic setting.

Future work includes the following.

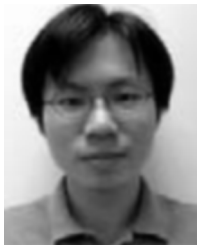
1) Apply existing Internet traffic trace in combination of real DDoS attack tool for a more complete evaluation of our scheme and others' schemes.
2) Consider the direction of distributed defense with router cooperation and communication.
3) Traceback.
4) Sophisticated scoring scheme with dynamic threshold.
5) Traffic detection based on individual router link.
6) Apply iceberg scheme to automatic identify profile.
7) Apply control network for defense router communication.

Acknowledgment

The authors would like to thank the reviewers for their useful comments that helped to improve the clarity and readability of this paper.

References

[1] W. Almesberger, Linux "Network traffic control—Implementation overview," 1999. [Online]. Available: http://diffserv.sourceforge.net/
[2] Traffic Management Specification ver. 4.1, (AF-TM-0121.000), 1999, The ATM Forum Technical Committee.
[3] B. Babcock, S Babu, M Datar, R Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. ACM Symp. Principles Datab. Syst.*, Jun/ 2002, pp. 1–16.
[4] J Barlow and W Thrower, "TFN2K—An analysis," Mar. 2003. [Online]. Available: http://ecurityresponse.Symantec.com/avcenter/security/Content/2000$_$02$_$10$_$a.html, (revision 1.3).
[5] S. M. Bellovin, M Leech, and T Taylor, "ICMP traceback messages," Internet Draft, draft-ietf-itrace-01.txt, Oct. 2001.
[6] D. J. Bernstein, "Syn cookies," 1996. [Online]. Available: http://cr.yp.to/syncookies.html
[7] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," in *Proc. Commun. ACM*, Jul. 1970, vol. 13, pp. 422–426.
[8] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey, 2002," in *Proc. 40th Annu. Allerton Conf. Commun., Control, Comput.*, 2002, pp. 636–646.
[9] H. Burch and B Cheswick, "Tracing anonymous packets to their approximate source," in *Proc. USENIX LISA*, 2000, pp. 319–328.
[10] E. Chan, "IDR: Intrusion detection router for defending distributed denial-of-service (DDoS) attack," in *Proc ISPAN*, May 2004, pp. 581–586.
[11] K. J. Houle, G. M. Weaver, N. Long, and R. Thomas, "Trends in denial of service attack technology," C.ERT/Coordination Center, 2001. [Online]. Available: http://www.cert.org/archive/pdf/DoS$_$trends.pdf
[12] Computer Security Institute and Federal Bureau of Investigation, "CSI/FBI computer crime and security survey," 2003. [Online]. Available: http://www.gocsi.com
[13] U.S. Department of Homeland Security, "The national strategy to security cyberspace," 2003. [Online]. Available: http://www.dhs.gov/interweb/assetlibrary/National_Cyberspace_Strategy.pdf
[14] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. SIGCOMM*, 2002, pp. 323–336.
[15] W. C. Feng, K. G. Shin, D. Kandlur, and D. Saha, "Stochastic fair blue: A queue management algorithm for enforcing fairness," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 1520–1529.
[16] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," IETF, RFC2827, May 2000.
[17] S Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Netw.*, vol. 3, no. 4, pp. 365–386, Aug. 1995.
[18] S. Floyd, "Notes of class-based queueing: Setting parameters," 1995. [Online]. Available: ftp://ftp.ee.lbl.gov/papers/params.ps.Z, Informal notes
[19] Chao Gong, Trinh Le, Turgay Korkmaz, and Kamil Sarac, "Single packet IP traceback in AS-level partial deployment scenario," in *Proc. IEEE GLOBECOM*, 2005, pp. 1817–1821.
[20] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003, pp. 99–110.
[21] V. acobson and M. J. Karels, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, 1988, pp. 314–329.
[22] H. Jiang and C. Dovrolis, "Source-level IP packets bursts: Causes and effects," in *Proc. ACM Internet Measurements Conf.*, Miami, FL, Oct. 2003, pp. 301–306.
[23] S. Jin and D. S. Yeung, IEEE Commun. Soc., 2004.
[24] Y. Kim, W. C. Lau, M. C. Chuah, and J. H. Chao, "PacketScore: Statistical-based overload control against distributed denial of service attacks," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 2594–2604.
[25] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and Bags," *ACM Trans. Datab. Syst.*, vol. 28, no. 1, pp. 51–55, Mar. 2003.
[26] C. Krügel, T. Toth, and E. Kirda, "Service specific anomaly detection for network intrusion detection," *ACM SAC*, pp. 201–208, 2002.
[27] [Online]. Available: http://www.tcpdump.org/
[28] D. Liu and F. Huebner, "Application profiling of ip traffic," in *Proc. 27th Annu. IEEE Conf. Local Comput. Netw.*, 2002, pp. 220–229.
[29] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," ACIRI & AT&T Labs Res., Tech. Rep., Jul. 2001.
[30] G. Marsaglia, "Xorshift RNGs," *J. Statistical Software*, vol. 8, no. 14, pp. 1–6, 2003.
[31] J. Mirkovic, P. Reiher, and G. Prier, "Attacking DDoS at the source," in *Proc. ICNP*, Nov. 2002, pp. 312–321.
[32] D. Moore, G. M. Voelker, and S. Savage, "Inferring Internet denial-of-service activity," in *Proc. 10th USENIX Security*, 2001, pp. 9–22.
[33] [Online]. Available: http://www-nrg.ee.lbl.gov/ns
[34] S. K. Rayanchu and G. Barua, "Tracing attackers with deterministic edge router marking (DERM)," in *Proc. 1st Int. Conf. Distrib. Comput. Internet Technol.*, 2004, pp. 49–52.
[35] S. K. Rayanchu, "Defending against slave and reflector attacks," in *Proc. Nat. Conf. Commun.*, 2005, Invited Lecture.
[36] P. R. Russell, "Unreliable guide to kernel hacking," Distributed with the 2.3.99 Linux RedHat Kernel, 2000.
[37] P. R. Russell and H. Welte, Linux netfilter Hacking HOWTO 2002. [Online]. Available: http://www.netfilter.org/
[38] J. H. Salim, R. Olsson, and A. Kuznetsov, "Beyond softnet," in *Proc. 5th Annu. Linux Showcase Conf.*, 2001, pp. 165–172.
[39] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *Proc. ACM SIGCOMM*, Aug. 2000, pp. 295–306.
[40] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based IP traceback," in *Proc. ACM SIGCOMM*, Aug. 2001, pp. 3–14.
[41] P. Vixie, G. Sneeringer, and M. Schleifer, "Event Report," Events of 21-Oct-2002 [Online]. Available: http://d.root-servers.org/october21.txt
[42] A. Yaar, A. Perrig, and D. Song, "Pi: A path identification mechanism to defend against DDoS attacks," in *Proc. IEEE Symp. Security Privacy*, 2003, pp. 93–107.
[43] D. K. Y. Yau, J. C. S. Lui, F. Liang, and Y. Yam, "Defending Against distributed denial-of-service attacks with max-min fair server-centric router throttles," *IEEE/ACM Trans. Netw.*, vol. 13, no. 1, pp. 29–42, 2005.



Eric Y. K. Chan received the M.Phil. degree in computer science from the University of Hong Kong.

He is currently an Assistant Computer Officer in the Department of Computer Science, University of Hong Kong. His research interests include information and network security.

H. W. Chan is an Assistant Professor in Department of Computer Science, University of Hong Kong.

His research interests include network security, mobile computing, and communication.

K. M. Chan is a Research Team Member at the University of Hong Kong.

P. S. Chan is a Research Team Member at the University of Hong Kong.

Samuel T. Chanson received the B.Eng. degree in electrical engineering from the University of Hong Kong and the M.Sc. and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley.

He was on the faculty of Purdue University and the University of British Columbia before he joined the Hong Kong University of Science and Technology (HKUST) in 1993. He was a full Professor of Computer Science and Director of Cyberspace Center at HKUST before he passed away in August 2005. His research interests include network protocols and network security.

M. H. Cheung is a Research Team Member at the University of Hong Kong.

C. F. Chong is a Lecturer in the Department of Computer Science, University of Hong Kong. His research interests are cryptanalysis, digital signature schemes, and elliptic curve cryptosystems.

K. P. Chow is an Associate Professor in the Department of Computer Science, University of Hong Kong, and is also the Associate Program Director for the M.Sc. in E-Commerce and Internet Computing Program at University of Hong Kong. His research interests are cryptography and software engineering.

Albert K. T. Hui is currently working towards the Ph.D. degree at the Department of Computer Science, Hong Kong University of Science and Technology. His research interests are in computer security.

Lucas C. K. Hui is the Founder and Honorary Director of the Center for Information Security and Cryptography, and concurrently an Associate Professor in the Department of Computer Science, University of Hong Kong. His research interests include information security, computer crime, cryptographic systems, and electronic commerce security.

S. K. Ip is a Research Team Member at the University of Hong Kong.

C. K. Lam is a Research Team Member at the University of Hong Kong.

W. C. Lau received the B.S.Eng. degree from the University of Hong Kong, Shatin, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Texas at Austin.

He is an Associate Professor with the Department of Information Engineering, Chinese University of Hong Kong (CUHK), Shatin, where he also serves as the Director of the Mobile Technologies Center (MobiTeC). From 1997 to 2004, he was with the Performance Analysis Department, Bell Laboratories, Lucent Technologies, Holmdel, NJ, where he served as a Performance Consultant and System Architect. Prior to joining CUHK, he was with Qualcomm, San Diego, CA, actively contributing to the design and standardization of IETF and 3G Mobility Management protocols and architecture. His research interest includes networking protocol design and performance analysis, traffic characterization, system modeling, and network security for high-speed wired and wireless networks.

K. H. Pun is an Associate Professor of Computer Science and a Honorary Associate Professor of Law in the University of Hong Kong. His research interests are legal aspects of computing: intellectual property protection of software, computer-related works.

Y. F. Tsang is a Research Team Member at the University of Hong Kong.

W. W. Tsang is an Associate Professor in the Department of Computer Science of the University of Hong Kong. His research interests are statistical computing and random number generation.

C. W. Tso received the Ph.D. degree from the Department of Computer Science, the University of Hong Kong.

D. Y. Yeung received the B.Eng. degree in electrical engineering and the M.Phil. degree in computer science from the University of Hong Kong, and the Ph.D. degree in computer science from the University of Southern California, Los Angeles.

He was an Assistant Professor at the Illinois Institute of Technology, Chicago before he joined the Department of Computer Science, Hong Kong University of Science and Technology, where he is currently an Associate Professor. He is currently on the Editorial Board of the *Journal of Artificial Intelligence Research* (JAIR). His research interests are in machine learning and pattern recognition, including both theory and applications.

S. M. Yiu is currently a Research Assistant Professor in the Department of Computer Science, University of Hong Kong.

His research interests include bioinformatics, information security, and cryptography.

K. Y. Yu received the Ph.D. degree from the Department of Computer Science, University of Hong Kong.

Weihua Ju was a Research Team Member at the University of Hong Kong.