ISBN: 978-93-83758-09-8

Distributed Denial-of-Service Attack Design Aspect of a Quantified Platform

[1]Pankaj Bajaj, [2]Anish Shandilya
[1,2] Assistant Professor, Geeta Engineering College, Panipat

[1]pankajbajaj4u@gmail.com, [2]shandilyaanish@gmail.com

Abstract: Distributed Denial-of-Service (DDoS) attacks are a more and more common problem facing Internet-exposed organizations today. One of the greatest challenges in the design and quantitative measurement of a platform for mitigating DDoS attacks is the ability to generate the required traffic patterns and loads required in a real network setup so as to notice performance impacts of modifications made. This paper presents the methodology used to test the network packet processing performance of the beginnings of a solution to the DDoS problem. The design overview of a custom packet generating solution is presented. The packet processing performance effects of various custom kernel options and a couple of modifications to the packet processing path in a popular Open-Source Operating System are considered.

I. INTRODUCTION TO DDOS: DEFINITION & RELEVANCE

Distributed Denial-of-Service (DDoS) attacks have become more and more prevalent throughout the past five years as high-speed Internet connectivity has become widespread. A DDoS attack is defined as follows: Multiple attack sites overwhelming a victim site with a large quantity of traffic. Victim's resources are exhausted such that legitimate clients are refused service. [Mirkovic 2003] It is important to note from the above definition that the victim's resources include not only available network bandwidth toward the victim's site but also the memory resources allocated to service said malicious traffic along the path to the attack target host (typically an end-host) and the processing delays caused throughout. Figure 1 below shows a DDoS attack directed at a target victim located on the public Internet within the local victim site topology shown.

It should be observed that the victim site is multi-homed by two connectivity providers: Connection Provider A and Connection Provider B. Equally noteworthy is that the depicted DDoS attack is coordinated by a single master host and 5 attack drone hosts, but that actual DDoS attacks can and often do involve thousands if not hundreds of thousands of attack hosts.

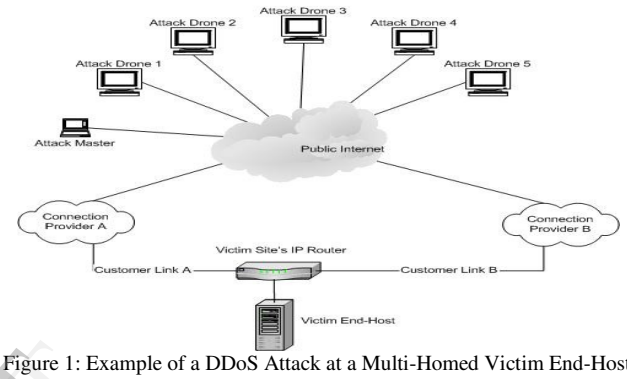

Figure 1: Example of a DDoS Attack at a Multi-Homed Victim End-Host

The attack depicted in Figure 1 has as ultimate objective the prevention of legitimate traffic flow to and from the Victim End-Host. Such a situation is achieved either by overwhelming or exploiting resources at the End-Host itself or by exhausting all available bandwidth in both Customer Link A and Customer Link B. Although the fact that the victim's site is multi-homed may act in the victim's favor, sufficiently strong attacks will lead to at least one of the two available links becoming exhausted, at which point traffic shall overflow to the remaining link thereby exhausting it as well. Descriptions and taxonomy of popular DDoS attacks and some mitigation techniques can be found in [Mirkovic 2003] and [UCLATr020018]. The seriousness of DDoS attacks should not be underestimated. The attacks are virtually unstoppable and are therefore currently the single largest threat to Internet-exposed organizations, often capable of bringing them offline for extended periods of time. This is evidenced by the following snippet from a front-page article in The Financial Times:

"More than a dozen offshore gambling sites serving the US market were hit by the so-called Distributed Denial of Service (DDoS) attacks and extortion demands in September. Sites have been asked to pay up to $50,000 to ensure they are free from attacks for a year. Police are urging victims not to give in to blackmail and to report the crime." (source: [FTIMES])

II. REQUIREMENTS FOR EFFECTIVELY MITIGATING DDOS

ISBN: 978-93-83758-09-8

Currently a variety of techniques are being employed in hopes of mitigating DDoS attacks. There are two distinct fronts on which these approaches attempt to combat the attack: the first is the source-end, and the second is at the victim (customer) site.

One common, source-end solution is the IP source-routing tactic being deployed by Internet Service Providers (ISPs) and traffic carriers. Since carriers typically have a large routing setup, they are able to easily examine packets before routing them to the public Internet. This is precisely the idea behind source routing: each packet attempting to leave the ISP's network has its source IP address examined. If the address is deemed to have originated from one of the carrier's own networks it is forwarded normally; however, if it is determined that the source IP address is not from within any of the ISP's networks the packet is dropped and will never reach its target. In the case of ordinary Denial of Service attacks the attacker will often make use of IP spoofing: the falsification of an IP packet's source address. In such instances Source Routing is very effective. However, in dealing with Distributed DoS attacks this tactic is essentially useless as IP spoofing is rarely necessary (the attack traffic is originating from otherwise legitimate hosts, often unbeknownst to their owners). When the overwhelming amount of attack traffic reaches the routers it will be recognized as legitimate traffic and promptly forwarded on its way toward the victim site. For this reason the detection of DDoS attacks is very difficult at the source. The difficulty of proper source-end detection is evidenced by much active research in this area (see for example [Mirkovic 2003]).

It is understandable that historically most solutions are placed at the point where the attack has its most profound impact: the victim site. Solutions that take the approaches of attempting to mitigate DDoS attacks at the victim site often involve employing sophisticated firewalls and packet filtering techniques. Such techniques are in theory extremely effective; however, in practice other bottlenecks, such as processing overhead, cause the victim site to drop legitimate traffic. Furthermore this type of approach mostly transfers the point of failure from the victim site itself to the point of the attempted solution. Figure 2 below illustrates the typical topology of these types of solutions. Through careful analysis it is realized that this approach is adheres to this transfer of failure point and is in fact inadequate.

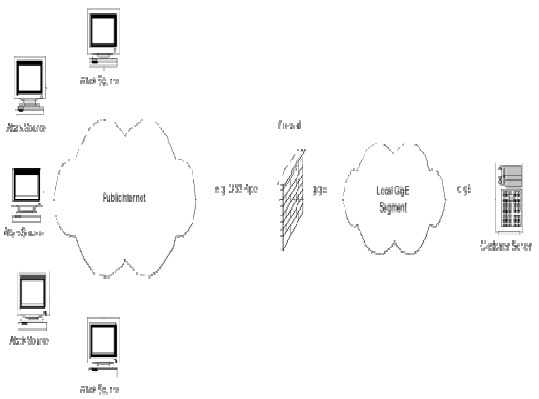

Figure 2: Typical Customer-End Firewall Topology

The first downfall of the situation depicted in Figure 2 begins at the firewall itself. As the DDoS attack rapidly propagates its massive amount of traffic, the firewall becomes overwhelmed due to the packet load. In this case the firewall itself becomes the bottleneck leading to failure: here the firewall is unable to examine the large number of arriving packets fast enough, causing an unmanageable build-up of traffic. As the build-up becomes unbearable to the victim site, its resources are rapidly consumed. The incredible amount of illegitimate traffic surged upon the victim has rendered the firewall essentially useless, successfully monopolizing resources.

The second breakdown of the type of solution modeled by Figure 2 arises along the pipe connecting the firewall to the public Internet. In the figure a DS3 line is depicted representing this potential tunnel of disaster. The impending failure that arises in these pipes during DDoS attacks occurs when they quickly fill due to aggregate traffic. In this case, the build-up propagates along this pipe all the way to routers injecting the traffic into the pipe. The queues of these routers rapidly fill due to their inability to send traffic along the filled pipes and as a consequence they begin to drop packets. Once again the DDoS attack has succeeded, and legitimate traffic destined for the victim site is refused service.

While it would seem that the world is doomed to live in fear of DDoS attacks forever, there is actually some active research in the field of DDoS mitigation that is displaying some impressive potential. The solutions proposed in these projects focus on filtering in firewalls. Here remarkable distributed and learning detection algorithms are created; these algorithms aim to discern legitimate from illegitimate traffic in real-time. Research of this variety is not only a step in the right direction, it is very important to the ultimate creation of a DDoS mitigation solution. One of the most important attributes of such a solution will undoubtedly be its ability to differentiate legitimate traffic from illegitimate malicious traffic. It is also important to remember that there are many different types of DDoS attacks, of which very few can be handled solely by the victim. This fact exerts the need for a distributed and possibly coordinated solution (such as [DWARD]). Furthermore, research of this type is especially interesting from an academic standpoint. This mainly attributed to the esteem of the fancy algorithms and fancy database structures produced.
One such detection and filtering solution is proposed by the D-WARD project:D-WARD (DDoS Network Attack Recognition and Defense) is a DDoS defense system deployed at a source-end network. Its Goal is twofold:

1. Detect outgoing DDoS attacks and stop them by controlling outgoing traffic to the victim.
2. Provide good service to legitimate transactions between the deploying network and the victim while the attack is on-going.

([DWARD])
The D-WARD approach is much like most other source-end solutions in that it is installed at source routers, in between the attack deploying network and the rest of the Internet. In fact the ultimate goal of D-WARD is to have its firewalls installed at a large number of such gateways. Once this is instituted, distributed algorithms involving many D-WARD firewalls are

ISBN: 978-93-83758-09-8

able to collaborate on filter rule-sets. The theoretical aspects of the D-WARD solution are seemingly invaluable; however, from a real world perspective source-end solutions are difficult to sell. Sadly, there is little incentive for carriers to widely deploy source-end solutions that have little immediate benefit for their own customers (see [DDoSChain] for a proposed economic solution to this incentive chain problem).

Although D-WARD and other areas of academic research have shown encouraging results, they all fail to address an important aspect of an effective DDoS mitigation solution: packet processing. The fact is that if packet processing is slow, then firewall and filtering mitigation methods are useless. This is mainly attributed to the fact that if the packets are being passed to the firewall at a rate higher than the firewall is able to process them, the packet processing becomes the bottleneck; in this case all efforts towards improving the firewall's filtering and inspection techniques are rendered practically useless.

It is evident that for a realistic DDoS-mitigating solution two distinct high-level requirements are needed. The first is the ability to detect and discern illegitimate traffic and subsequently install firewall rules/filters to block it. This is the type of work being actively researched in solutions such as D-WARD. The second aspect is the ability to process packets quickly and efficiently match them against large number of rules. The research presented in the remainder of this paper focuses directly on this second challenge.

III. CHALLENGES IN MEASURING NETWORK PACKET PROCESSING PERFORMANCE

It is often thought that hardware offloading should solve the packet processing bottleneck; however, in the case of DDoS attacks, hardware offloading is not as useful as in High-End IP Router design (see [Juniper] high-end IP router architecture). In general ASICs and hardware offloading do not work well for firewalls, particularly those that aim to protect against DDoS attacks. DDoS attacks can be composed of many different types of traffic, some of which require deep packet inspection to filter, and others which can be filtered merely with shallow network-layer header inspection. This conflicting pair of requirements is why it is difficult in anti-DDoS firewall design to find a common case to offload to hardware. Essentially one would need to build the features and flexibility of the operating system and software-based firewall into the hardware to appropriately deal with varying types of packets. This need in turn causes the hardware requirements for the offloading unit to increase thereby invalidating the hardware-offloading design pattern. A firewall that could successfully offload network-layer header inspection to hardware could therefore be rendered useless by crafting an attack that cannot be filtered with merely network-layer header inspection.

With this in mind, it is more natural to consider an off-the-shelf server-grade shared-memory multi-processor (SMP) system as a platform for a DDoS-mitigating firewall, rather than a network device with custom circuitry. For this purpose, the open-source FreeBSD operating system which has ongoing and active development for SMP hardware was an obvious choice (refer to [FreeBSD] and [Lehey] for more information).

The remainder of this paper considers two fundamental challenges in packet processing performance improvement for the beginnings of a DDoS-mitigating solution:

(1) Determining the status-quo packet processing performance of the FreeBSD operating system on SMP hardware;
(2) Considering an experimental modification to the FreeBSD development-branch kernel (operating system core software) and gauging its impact on packet processing.

In order to accurately accomplish the primary goal of measuring the FreeBSD system's current packet processing abilities, the following was required:

(1) Set up or make use of a test network environment in which sufficient traffic load could be generated so as to stress a target FreeBSD SMP system's packet processing code paths;
(2) Ensure that the target FreeBSD SMP system's hardware is not the limiting factor in testing;
(3) Ensure that the packet generation infrastructure (i.e., packet source hosts) within the test network are capable of generating sufficiently high packet rates without bottlenecking on the wire-speed limits of the test network and that flexible traffic patterns can be easily generated.

Requirements (1) and (2) are discussed in this section while requirement (3) is addressed in the following section entitled Design & Implementation of a Packet Factory & Generator Software Solution.

The test cluster used for performing packet processing testing and measurements for the work presented herein belongs to the FreeBSD Project and is commonly referred to as the Zoo test bed or cluster (see [ZooCluster] for more information). The current topology of the Zoo test cluster is depicted in Figure 3.

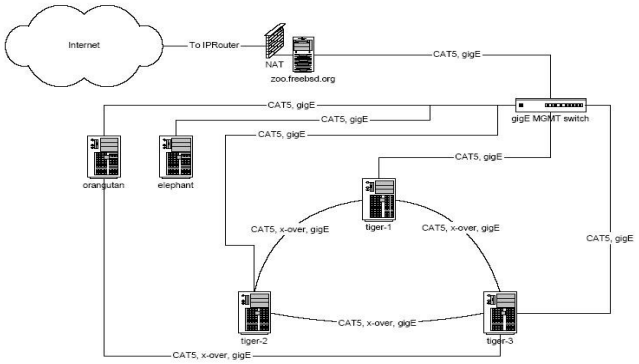

Figure 3: Current Zoo Test Cluster Network Topology

While various different machines are pictured in Figure 3, only the 3 identical tiger machines were used in packet processing performance data gathering. The tiger machines consist of three hardware-identical SMP servers with four gigabit Ethernet interfaces each, such that one gigabit Ethernet interface is connected to a management switch ("MGMT switch" in Figure 3) thereby contributing to the internal management and configuration network for all Zoo cluster machines. Of the remaining three network interfaces on each tiger, two are used to directly connect to each of the remaining two tiger hosts. Therefore, all three tiger machines are completely directly interconnected thereby forming a gigabit Ethernet loop, ensuring that no switching elements exist

ISBN: 978-93-83758-09-8

between any hosts and eliminating the possibility that a switching element becomes the bottleneck.

As mentioned, the three tiger hosts used for testing are all SMP-capable servers and are equipped with identical server-grade hardware. Each tiger machine features dual Intel Xeon 3.06GHz Hyperthreading (HTT)-capable CPUs on an Intel Server Board SE7501wv2 which includes three PCI buses and a 533MHz front-side-bus (see [Intel] for more information). Each machine has 4GB of RAM. It is important to note that the Intel SE7501wv2 board has a built-in Intel 82546EB dual-port gigabit Ethernet controller and that the machines were also equipped with a second dual-port 82546EB in the form of a Low-Profile 64-bit PCI-X card. The latter fact is important to remember because it ensures that each 82546EB sits on a separate uncontested 64-bit PCI bus, thereby eliminating the possibility of bottlenecking in hardware during high-frequency 82546EB-to-CPU or RAM transfers which occur as a result of high packet rates to which the tiger machines will be subjected to.

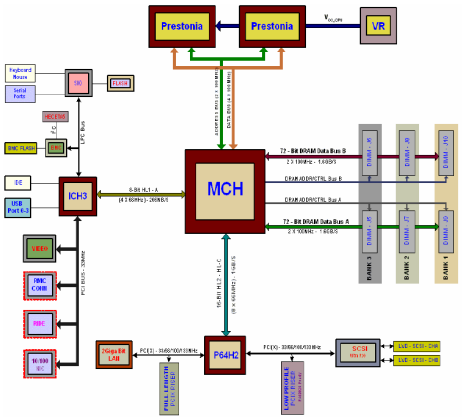

Figure 4: Intel Server Board SE7501wv2 Block Diagram (Source: [IntelTPS])

The bottom of Figure 4 depicts the P64H2 on the Intel SE7501wv2 board being connected via PCI-X to the dual-port on-board 82546EB (on the left) and the Full-Profile PCI-X slot. It also shows that the P64H2 is connected (on the right) to the Low-Profile PCI-X slot and the SCSI Ultra320 controller found on some versions of the board. The versions of the Intel SE7501wv2 found in the tiger machines do not include the on-board SCSI Ultra320 controller and so, in fact, the right-side connection to the P64H2 shown in Figure 4 only connects the Low-Profile PCI-X slot. In the tiger machines, the Full-Profile PCI-X slot is left empty while the Low-Profile slot sports the PCI-X card version of the dual-port 82546EB. Supposing that the PCI-X buses to the P64H2 are clocked at 100MHz, and for the moment neglecting PCI-X bus acquisition overhead, the amount of data transferable via the 64-bit wide bus per second is:

$$\frac{100000000}{s} \bullet 64bits = \frac{6400000000bits}{s}$$

Since the 82546EB controllers are dual-port, they require at worse 2Gbits bus bandwidth (neglecting required PCI-X bus acquisition overhead). The bus itself, since uncontested, can provide approximately 6.4Gbps. The difference, 4.4Gbps, is more than sufficient to handle bus acquisition overhead when the bus is uncontested, even in the most conservative of guesses (see [PCIX] for justification). The fact that the 82546EB controllers perform significant traffic coalescing (see [82546EB]) further ensures that not every packet results in a bus transaction, particularly at high packet rates, thereby lowering the associated and neglected overhead.

IV. CONCLUSIONS ON EXPERIMENTAL MODIFICATIONS TO THE FREEBSD PACKET PROCESSING CODE PATHS

Two experimental changes critically impacting the FreeBSD receive packet processing code paths were considered. The first change is referred to as the UMA critical sections optimization whereas the second is a modification of an existing but highly-experimental mechanism in FreeBSD 6.0-CURRENT involving the net.isr.enable run-time tunable system control variable. The first of the considered changes was successfully benchmarked with respect to the three metrics defined in the Measuring Status Quo FreeBSD Packet Processing Performance section of this paper. The second change causes a live-lock phenomenon to occur on the receiver (sink) host, tiger2, and so performance data was not successfully collected. Work is ongoing to eliminate the live-lock condition and some details are presented below.

In order to understand the potential impact of the UMA critical section optimization some background knowledge is required. The UMA subsystem is a general multi-processor optimized memory allocator derived from the classic Slab memory allocator design (see [Bonwick]). The UMA allocator is found in FreeBSD 6.0-CURRENT and features small higher-layer per-CPU caches in order to provide a fast-path for common case allocations. Simply, if memory buffers can be found in the executing CPU's private UMA cache at allocation time, it will be immediately allocated. Otherwise, a global memory buffer cache will need to be checked followed by a call into the lower-level Virtual Memory (VM) subsystem code should the latter also be found empty.

In the current FreeBSD 6.0-CURRENT implementation, per-CPU mutual exclusion locks protect the per-CPU UMA caches. This has the benefit of ensuring synchronized access to the per-CPU UMA cache should preemption occur at the time of access. The unfortunate side-effect of using per-CPU mutual exclusion locks is their lousy performance, even in the common case where the lock is uncontested (see [McKenney] for a general alternative to mutual exclusion applicable to particular scenarios for this very reason). Currently, all network buffers (as well as most other kernel data structures or buffers) in FreeBSD 6.0-CURRENT are allocated using the UMA allocator.

The UMA critical section optimization replaces the UMA per-CPU cache mutual exclusion locks with an micro-optimized interrupt masking mechanism along with temporary CPU thread pin-down (this consists of asking the kernel scheduler to not migrate the currently executing thread from the currently executing CPU in the case of preemption). The argument for the change is that optimized interrupt masking and temporary CPU thread pin-down are operations significantly cheaper than mutual lock acquisitions in the common case (refer to [Hlusi]

ISBN: 978-93-83758-09-8

for a discussion on relative cost of SMP-related operations on contemporary Intel hardware).

Figures 26, 27, and 28 depict the performance of the UMA critical section optimization change and the VANILLA 6.0-CURRENT kernel with respect to the three previously defined metrics. As depicted, input processing appears to be more regular (smoother curve) with the UMA critical section optimization than without, although input performance is slightly worse. On the other hand, the UMA critical section optimization shows significantly fewer input interface errors as well as significantly higher (500% on average) output processing performance. Due to the high sensitivity of the recorded metrics to slight nuances in scheduling behavior and the nature of the optimizations, the results are in fact not surprising.

The UMA critical section optimization makes use of temporary preemption disabling so as to ensure that an incoming interrupt does not cause UMA per-CPU cache corruption during cache access. Although the argument for the change is that the net common-case cost of an optimized temporary preemption disable accompanied by a temporary CPU thread pin-down is lower than the common-case cost of a mutual exclusion lock acquisition, the effect reported by the metrics in Figures is actually due to the change's impact on scheduling behavior. The temporary preemption disable consists of interrupt masking, thereby causing a short-term priority inversion to be performed by a low priority thread. This is in fact what happens during the tests. Namely, the receiver (sink) is subjected to a very high interrupt load. The more interrupts the sink is able to process, the higher the input processing numbers will be, but so will the input interface errors. As well, output processing numbers will be lower. The reason for this inverse relationship is that higher interrupt processing starves out available CPU time by preventing CPUs from processing any lower priority threads. In the case of the UMA critical section optimizations, the temporary priority inversions prevent the high number of incoming interrupts from monopolizing the processor, causing input performance to drop but output performance and the netISR thread's performance to increase (this explains the lower number of input interface errors).

V. REFERENCES

[1] [Mirkovic 2003] J. Mirkovic, G. Prier and P. Reiher, Challenges of Source-End DDoS Defense, Proceedings of 2nd IEEE International Symposium on Network Computing and Applications, April 2003.

[2] [UCLATr020018] J. Mirkovic, J. Martin and P. Reiher, A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms, UCLA CSD Technical Report CSD-TR-020018.

[3] [DWARD] J. Mirkovic, D-WARD: Source-End Defense Against Distributed Denial-of-Service Attacks, Ph.D. Thesis, UCLA, August 2003

[4] [FreeBSD] [WEB] The FreeBSD Project website. URL: http://www.FreeBSD.org/

[5] [Netperf] [WEB] The Public Netperf Performance Tool website. URL:http://www.netperf.org/netperf/NetperfPage.html

[6] [NetPIPE] [WEB] A Network Protocol Independent Performance Evaluator: NetPIPE Project website. URL: http://www.scl.ameslab.gov/netpipe/

[7] [ttcp] [WEB] PCAUSA Test TCP Benchmarking Tool for Measuring TCP and UDP Performance (Project website). URL: http://www.pcausa.com/Utilities/pcattcp.htm

[8] [ZooCluster] [WEB] The Zoo Test Cluster News and Updates website. URL: http://zoo.unixdaemons.com/

[9] [Trinoo] [WEB] David Dittrich. The DoS Project's "Trinoo" Distributed Denial of Service Attack Tool. University of Washington public document. October 1999.URL: http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt

[10] [TFN] [WEB] David Dittrich. The "Tribe Flood Network" Distributed Denial of Service Attack Tool. University of Washington public document. October, 1999.URL: http://staff.washington.edu/dittrich/misc/tfn.analysis.txt

[11] [DDoSTools] [WEB] Advanced Networking Management Lab Distributed Denial of Service Attack Tools. Public document from Pervasive Technology Labs at Indiana University. URL:http://www.anml.iu.edu/ddos/tools.html