The Internet Protocol Journal

December 2004 Volume 7, Number 4

A Quarterly Technical Publication for Internet and Intranet Professionals

In This Issue

F r o m T h e E d i t o r

The electronics industry is full of examples of devices which contain one or two "special-purpose" chips. Your computer probably has a modem that is implemented with a single chip and a few analog components. It probably also contains a dedicated graphics processor responsible for driving your display. In networking, vendors have long since realized that in order to design highly efficient routers or switches, a custom-designed *network processor* is a good solution. We asked Doug Comer to give us an overview of network processors.

Attacks against individual computers on a network have become all too common. Usually these attacks take the form of a virus or worm which arrives via e-mail to the victim's machine. The industry has been relatively quick in responding to such attacks by means of antivirus software, as well as sophisticated filtering of content "on the way in." A more serious form of attack is the *Distributed Denial-of-Service* (DDoS) attack which may render an entire network unusable. Charalampos Patrikakis, Michalis Masikos, and Olga Zouraraki give an overview of the many variants of denial-of-service attacks and what can be done to prevent them.

Although we make every effort to provide you with an error-free journal, mistakes do happen occasionally. Sometimes it takes careful analysis by a reader to spot the mistake, and we are grateful for the correction provided in the "Letter to the Editor" on page 36. Other times, technology just gets in our way, such as when all the non-printing end-of-line and TAB characters became very much "printing"—see page 35 of the printed version of Volume 7, No. 3. At least it didn't show up in the PDF or HTML versions.

Take a moment to visit our Website: **http://www.cisco.com/ipj** and update your mailing address if necessary. You will also find all back issues and index files at the same address.

—Ole J. Jacobsen, Editor and Publisher
ole@cisco.com

Network Processors: Programmable Technology for Building Network Systems

by Douglas Comer, Cisco Systems (on leave from Purdue University)

Chip vendors have defined a new technology that can be used to implement packet-processing systems such as routers, switches, and firewalls. The technology offers the advantages of being software-programmable and sufficiently high-speed to accommodate interfaces running at 10 Gbps.

This article provides an overview of the technology, describes the motivations, and presents a brief survey of hardware architectures. It also discusses the relationship between programming and the underlying hardware.

A wide variety of packet-processing systems are used in the Internet, including DSL modems, Ethernet switches, IP routers, *Network Address Translation* (NAT) boxes, *Intrusion Detection Systems* (IDS), Softswitches used for *Voice over IP* (VoIP), and security firewalls. Such systems are engineered to provide maximal functionality and performance (for example, operate at wire speed) while meeting constraints on size, cost, and time to market.

Engineers who design network systems face the additional challenges of keeping designs scalable, general, and flexible. In particular, because industry trends change rapidly, typical engineering efforts must accommodate changes in requirements during product construction and changes in the specification for a next-generation product.

Generations of Network Systems

During the past 20 years, engineering of network systems has changed dramatically. Architectures can be divided broadly into three generations:

- *First generation* (circa 1980s): Software running on a standard processor (for example, an IP router built by adding software to a standard minicomputer),
- *Second generation* (mid 1990s): Classification and a few other functions offloaded from the CPU with special-purpose hardware, and a higher-speed switching fabric replacing a shared bus.
- *Third generation* (late 1990s): Completely decentralized design with *Application-Specific Integrated Circuit* (ASIC) hardware plus a dedicated processor on each network interface offloading the CPU and handling the fast data path.

The change from a centralized to a completely distributed architecture has been fundamental because it introduces additional complexity. For example, in a third-generation IP router, where each network interface has a copy of the routing table, changing routes is difficult because all copies must be coordinated to ensure correctness and the router should not stop processing packets while changes are propagated.

Motivation for Network Processors

Although the demand for speed pushed engineers to use ASIC hardware in third-generation designs, the results were disappointing. First, building an ASIC costs approximately US$1 million. Second, it takes 18 to 22 months to generate a working ASIC chip. Third, although engineers can use software simulators to test ASIC designs before chips are manufactured, networking tasks are so complex that simulators cannot handle the thousands of packet sequences needed to verify the functionality. Fourth, and most important, ASICs are inflexible.

The inflexibility of ASICs impacts network systems design in two ways. First, changes during construction can cause substantial delay because a small change in requirements can require massive changes in the chip layout. Second, adapting an ASIC for use in another product or the next version of the current project can introduce high cost and long delays. Typically, a silicon respin takes an additional 18 to 20 months.

Network-Processor Technology

In the late 1990s as demand for rapid changes in network systems increased, chip manufacturers began to explore a new approach: programmable processors designed specifically for packet-processing tasks. The goal was clear: combine the advantage of software programmability, the hallmark of the first-generation network systems, with high speed, the hallmark of third-generation network systems.

Chip vendors named the new technology *network processors*, and predicted that in the future, most network systems would be constructed using network processors. Of course, before the prediction could come true, vendors faced a tough challenge: programming introduces an extra level of indirection, meaning that functionality implemented directly in hardware always performs faster than the same functionality implemented with software. Thus, to make a network processor fast enough, packet-processing tasks need to be identified and special-purpose hardware units constructed to handle the most intensive tasks.

Interestingly, vendors also face an economic challenge: although an ASIC costs a million dollars to produce, subsequent copies of the chip can be manufactured at very low cost. Thus, the initial development cost can be amortized over many copies. In contrast, purchasing conventional processors does not entail any initial development cost, but vendors typically charge at least an order of magnitude more per unit than for copies of an ASIC. So, vendors must consider a pricing strategy that entices systems builders to use network processors in systems that have many network interfaces with multiple processors per interface.

A Plethora of Architectures

As vendors began to create network processors, fundamental questions arose. What are the most important protocol-processing tasks to optimize? What hardware units should a network processor provide to increase performance? What I/O interfaces are needed? What sizes of instruction store and data store are needed? What memory technologies should be used (for example, *Static Random-Access Memory* [SRAM], *Dynamic Random-Access Memory* [DRAM], or others)? How should functional units on the network-processor chip be organized and interconnected (for example, what on-chip bus infrastructure should be used)?

Interestingly, although they realized that it was essential to identify the basic protocol-processing tasks before hardware could be built to handle those tasks efficiently, chip vendors had little help from the research community. Much effort had been expended considering how to implement specific protocols such as IP or TCP on conventional processors. However, researchers had not considered building blocks that worked across all types of network systems and all layers of the protocol stack. Consequently, in addition to designing network-processor chips, vendors needed to decide which protocol functions to embed in hardware, which to make programmable, and which (if any) to leave for special-purpose interface chips or coprocessors. Finally, chip vendors needed to choose software support including programming language(s), compilers, assemblers, linkers, loaders, libraries, and reference implementations.

Faced with a myriad of questions and possibilities about how to design network processors and the recognition that potential revenue was high if a design became successful, chip vendors reacted in the expected way: each vendor generated a design and presented it to the engineering community. By January 2003, more than 30 chip vendors sold products under the label “network processor.”

Unfortunately, the euphoria did not last, and many designs did not receive wide acceptance. Thus, companies began to withdraw from the network-processor market, and by January 2004, fewer than 30 companies sold network processors.

Basic Architectural Approaches

Hardware engineers use three basic techniques to achieve high-speed processing: a single processor with a fast clock rate, parallel processors, and hardware pipelining. Figure 1 illustrates packet flow through a single processor, which is known as an *embedded processor architecture* or a *run-to-completion model*. In the figure, three functions must be performed on each packet.

Figure 1: Embedded Processor Architecture in Which a Single Processor Handles All Packets

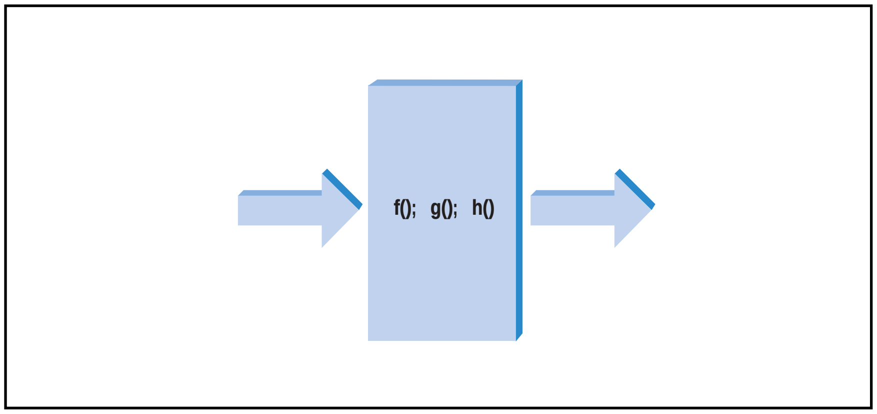


Figure 2 illustrates packet flow through an architecture that uses a parallel approach. A coordination mechanism on the ingress side chooses which packets are sent to which processor. Coordination hardware can use a simplistic round-robin approach in which a processor receives every *N*th packet, or a sophisticated approach in which a processor receives a packet whenever the processor becomes idle.

Figure 2: Parallel Architecture in Which the Incoming Packet Flow Is Divided Among Multiple Processors

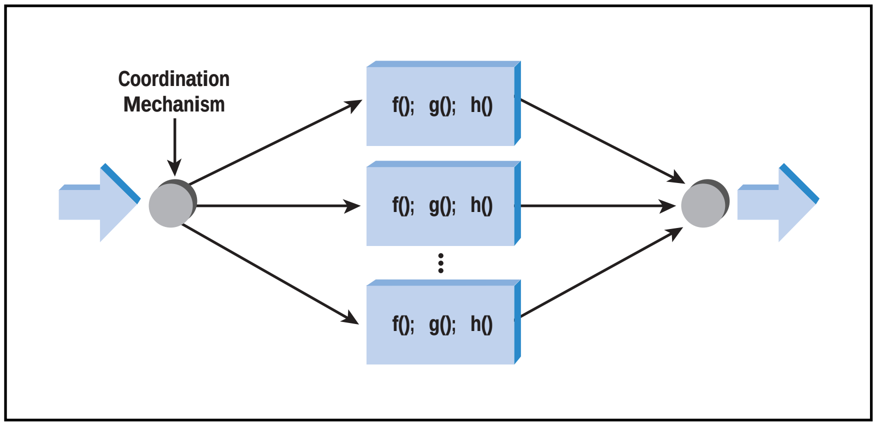
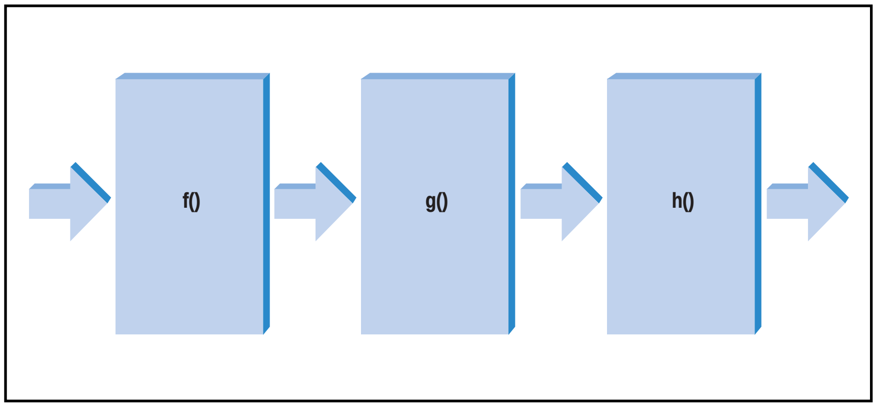


Figure 3 illustrates packet flow through a pipeline architecture. Each packet flows through the entire pipeline, and a given stage of the pipeline performs part of the required processing.

Figure 3: Pipeline Architecture in Which Each Incoming Packet Flows Through Multiple Stages of a Pipeline



As we will see, pipelining and parallelism can be combined to produce hybrid designs. For example, it is possible to have a pipeline in which each individual stage is implemented by parallel processors or a parallel architecture in which each parallel unit is implemented with a pipeline.

Examples of Commercial Architectures

To appreciate the broad range of network-processor architectures, we will examine a few commercial examples. Commercial network processors first emerged in the late 1990s, and were used in products as early as 2000. The examples contained in this article are chosen to illustrate concepts and show broad categories, not to endorse particular vendors or products. Thus, the examples are not necessarily the best, nor the most current.

Augmented RISC (Alchemy)

The first example, from Alchemy Semiconductor (now owned by Advanced Micro Devices), illustrates an embedded processor augmented with special instructions and I/O interfaces.

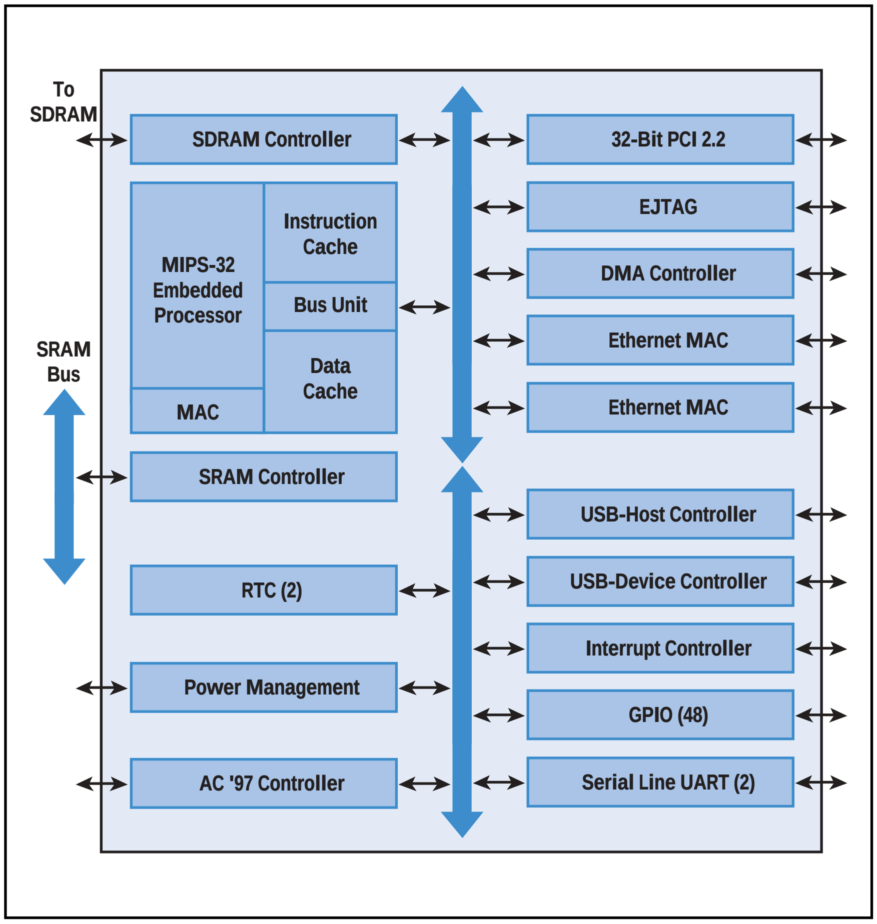
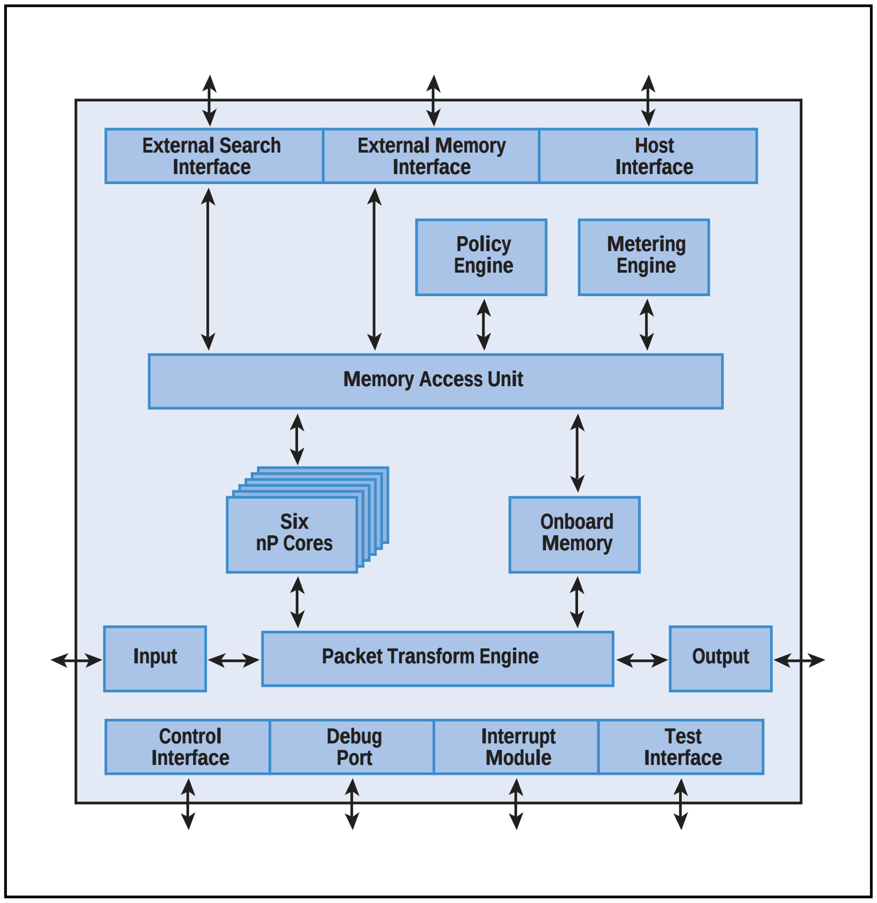


Figure 4: An Example Embedded Processor Architecture: The Processor Has Extra Instructions to Speed Packet Processing

Parallel Processors Plus Coprocessors (AMCC)

A network processor from AMCC uses an architecture with parallel processors plus coprocessors that handle packet-processing tasks. When a packet arrives, one of the parallel processors, called *cores*, handles the packet. The coprocessors are shared—any of the parallel processors can invoke a coprocessor, when needed.

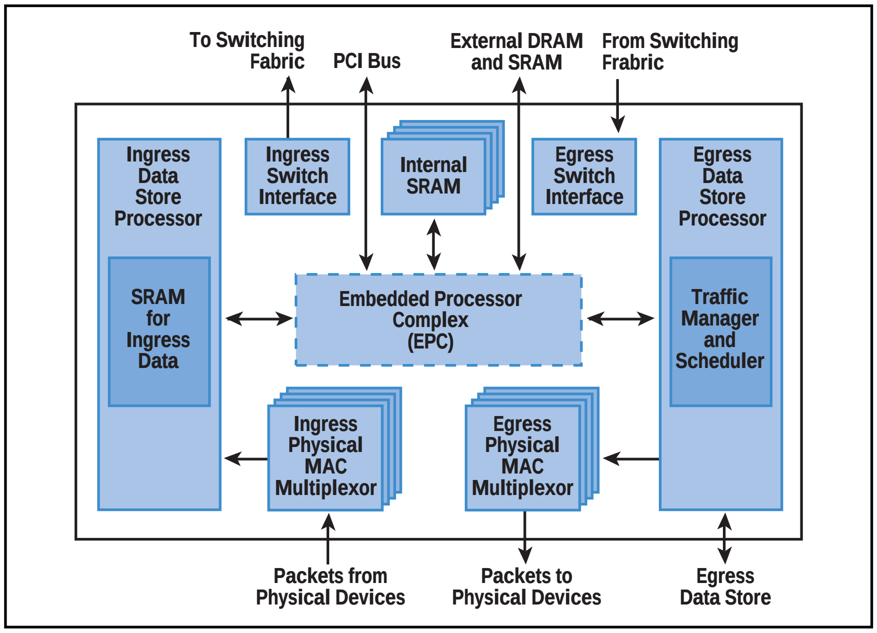
Figure 5: An Example Parallel Architecture that Uses Special-Purpose Coprocessors to Speed Execution



Extensive and Diverse Processors (Hifn)

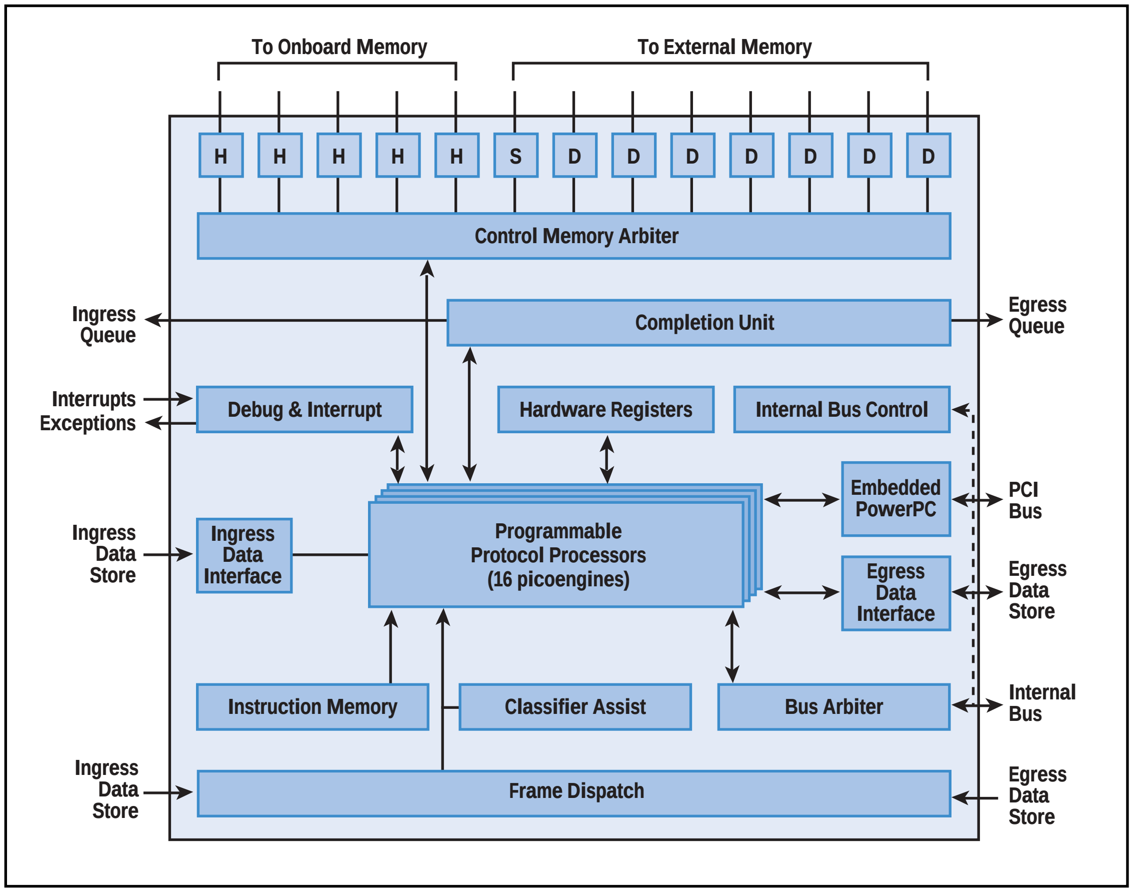
A network processor (named *Rainier*) originally developed by IBM and now owned by Hifn Corporation uses a parallel architecture, and includes a variety of special-purpose and general-purpose processors. For example, the chip provides parallel ingress and egress hardware to handle multiple high-speed network interfaces. It also has intelligent queue-management hardware that enqueues incoming packets in an ingress data store, a switching fabric interface built onto the chip, and an intelligent egress data store. Figure 6 illustrates the overall architecture of the Hifn chip.

Figure 6: An Example Parallel Architecture that Includes Hardware Support for Ingress and Egress Processing as well as Intelligent Queuing


To Switching Fabric
PCI Bus
External DRAM and SRAM
From Switching Frabric
Ingress Data Store Processor
SRAM for Ingress Data
Ingress Switch Interface
Internal SRAM
Egress Switch Interface
Egress Data Store Processor
Traffic Manager and Scheduler
Embedded Processor Complex (EPC)
Ingress Physical MAC Multiplexor
Egress Physical MAC Multiplexor
Packets from Physical Devices
Packets to Physical Devices
Egress Data Store

The *Embedded Processor Complex* (EPC) on the Hifn chip contains 16 programmable packet processors, called *picoengines*, as well as various other coprocessors. In addition, the EPC contains an embedded PowerPC to handle control and management tasks. Figure 7 shows a few of the many processors in the EPC.

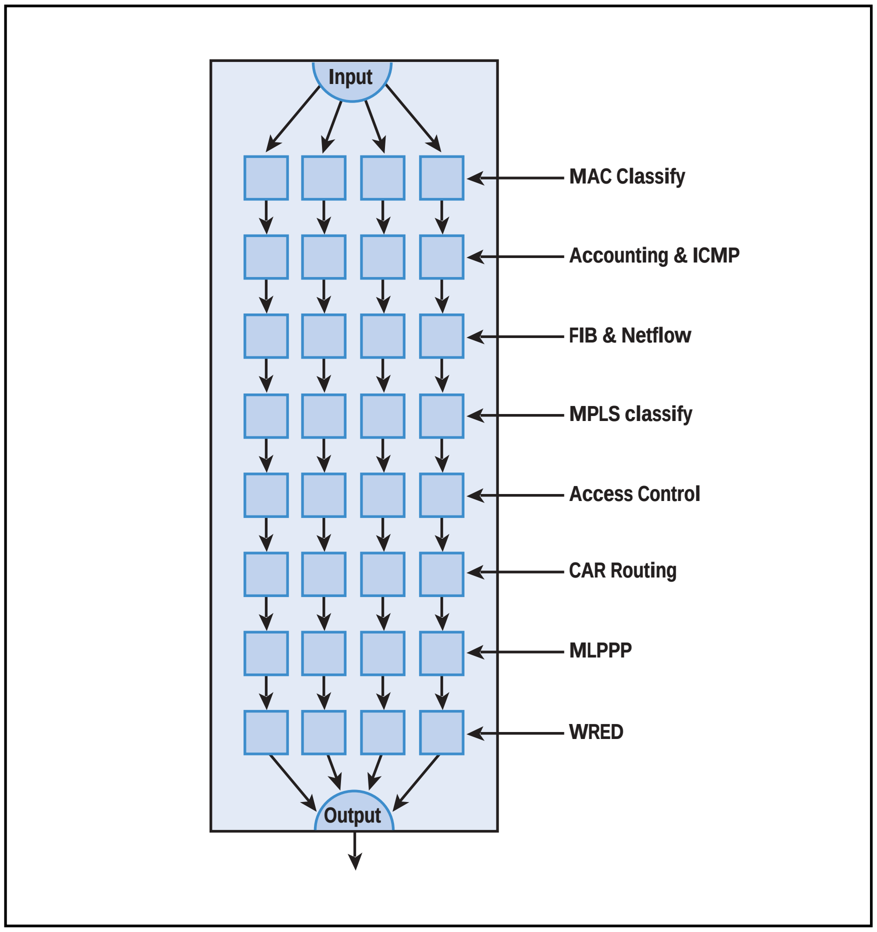
Figure 7: Structure of the Embedded Processor Complex on the Example Network Processor in Figure 6



Parallel Pipelines of Homogeneous Processors (Cisco)

Although it is not a chip vendor, Cisco Systems uses network processors in its products, and has developed network processors for internal use. One of the more interesting designs employs parallel pipelines of homogeneous processors. Figure 8 illustrates the architecture of the Cisco chip. When a packet enters, the hardware selects one of the pipelines, and the packet travels through the entire pipeline.

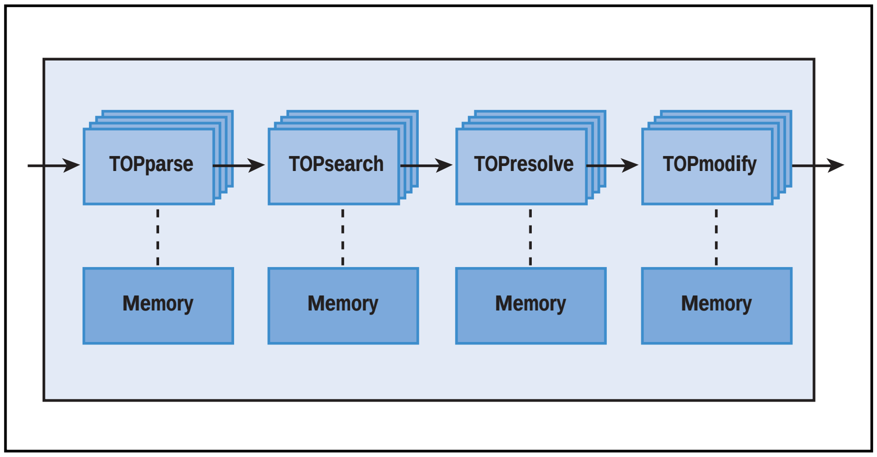
Figure 8: An Example Architecture that Uses Parallel Pipelines of Homogeneous Processors



Pipeline of Parallel Heterogeneous Processors (EZchip)

EZchip Corporation sells a network processor that combines pipelining and parallelism by using a four-stage pipeline in which each stage is implemented by parallel processors. However, instead of using the same processor type at each stage, the EZchip architecture employs heterogeneous processors, with the processor type at each stage optimized for a certain task (for example, the processor that runs forwarding code is optimized for table lookup). Figure 9 illustrates the architecture.

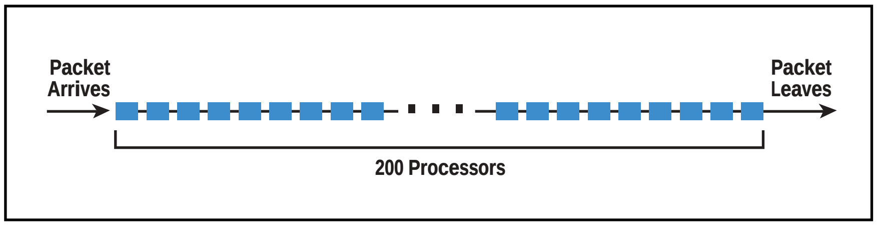
Figure 9: An Example Architecture that Uses a Pipeline of Parallel Stages with Heterogeneous Processors



Extremely Long Pipeline (Xelerated)

Xelerated Corporation sells an interesting network processor that uses a pipelining approach. Unlike other network processors, the Xelerated chip uses an extremely long pipeline of 200 stages. Figure 10 illustrates the overall architecture. To achieve high speed, each stage is limited to executing four instructions per packet.

Figure 10: An Example of an Extremely Long Pipeline with 200 Stages



In fact, the Xelerated architecture is more complex than the figure shows because the pipeline contains special hardware units after every 10 stages that allow external communication (for example, access to external memory or a call to a coprocessor).

More Details and Example Network-Processor Source Code

The previous survey is not meant to be complete. Two notable network processors have been omitted. Agere Systems and Intel each manufacture a network processor. Agere's design consists of a short pipeline that has two basic stages. Agere's architecture is both interesting and unusual because the two stages are composed of unconventional processors. For example, the processor used for classification performs high-speed pattern matching, but does not have conventional instructions for iteration or conditional testing. For details about the Agere network processor see[1], which includes the source code for an example *Differentiated Services* (DiffServ) network system.

Intel's chip uses a parallel approach in which a set of *microengines* are programmed to handle packets. The Intel hardware allows a programmer to pass packets between microengines, meaning a programmer can decide to arrange microengines in a software pipeline. For details about the Intel network processor see[2], which includes the source code for an example NAT implementation.

Programming Network Processors

Although the general idea of building programmable devices seems appealing, most network-processor designs make programming difficult. In particular, to achieve high speed, many designs use low-level hardware constructs and require a programmer to accommodate the hardware by writing low-level code. Many network processors are much closer to a microcontroller than a conventional processor, and are programmed in *microassembly* language. Programmers must be conscious of details such as register banks.

Programming is especially difficult in cases where the network-processor hardware uses explicit parallelism and requires a programmer to plan program execution in such a way that processors do not contend for resources simultaneously or otherwise stall. For example, on one vendor's chip, a packet processor can execute several hundred instructions while waiting for a single memory access to complete. Thus, to achieve high performance, a programmer must start a memory operation, go on with other calculations while the memory operation proceeds, and then check that the operation has completed.

In addition to considering processing, some network processors provide a set of memory technologies, and require a programmer to allocate each data item to a specific memory. A programmer must understand memory latency, the expected lifetime of a data object, and the expected frequency of access as well as properties of the hardware such as memory banks and interleaving.

A few pleasant exceptions exist. For example, Agere Systems provides special-purpose, high-level programming languages to program its network processors. Thus, it is easy to write classification code or traffic-management scripts for an Agere processor. More important, an Agere chip offers implicit parallelism: a programmer writes code as if a single processor is executing the program; the hardware automatically runs multiple copies on parallel hardware units and handles all details of coordination and synchronization.

Another pleasant exception comes from IP Fabrics, which has focused on building tools to simplify programming. Like Agere, IP Fabrics has developed a high-level language that allows a programmer to specify packet classification and the subsequent actions to be taken. The language from IP Fabrics is even more compact than the language from Agere.

Summary

To provide maximal flexibility, ease of change, and rapid development for network systems, chip vendors have defined a new technology known as network processors. The goal is to create chips for packet processing that combine the flexibility of programmable processors with the high speed of ASICs.

Because there is no consensus on which packet-processing functions are needed or which hardware architecture(s) are best, vendors have created many architectural experiments. The basic approaches comprise an embedded processor, parallelism, and hardware pipelining. Commercial chips often combine more than one approach (for example, a pipeline of parallel stages or parallel pipelines).

Programming network processors can be difficult because many network processors provide low-level hardware that requires a programmer to use a microassembly language and handle processor, memory, and parallelism details. A few exceptions exist where a vendor provides a high-level language.

References

[1] Comer, D., *Network Systems Design Using Network Processors, Agere Version*, Prentice Hall, 2005.

[2] Comer, D., *Network Systems Design Using Network Processors, Intel 2xxx Version*, Prentice Hall, 2005.

This article is based on material in *Network Systems Design Using Network Processors, Agere Version*, and *Network Systems Design Using Network Processors, Intel 2xxx Version* by Doug Comer. Both books are published by Prentice Hall in 2005. Used with permission.

DOUGLAS E. COMER is a Visiting Faculty at Cisco Systems, a Distinguished Professor of Computer Science at Purdue University, a Fellow of the ACM, and editor-in-chief of the journal *Software—Practice and Experience*. As a member of the IAB, he participated in the formation of the Internet, and is considered a leading authority on TCP/IP and Internetworking. He is the author of 16 technical books that that have been translated into 14 languages, and are used around the world in industry and academia. Comer has been working with network processors for several years, and has reference platforms from three leading vendors in his lab at Purdue. E-mail: **comer@cs.purdue.edu**

Distributed Denial of Service Attacks

*By Charalampos Patrikakis, Michalis Masikos, and Olga Zouraraki
National Technical University of Athens*

The Internet consists of hundreds of millions of computers distributed all around the world. Millions of people use the Internet daily, taking full advantage of the available services at both personal and professional levels. The interconnectivity among computers on which the World Wide Web relies, however, renders its nodes an easy target for malicious users who attempt to exhaust their resources and launch *Denial-of-Service* (DoS) attacks against them.

A DoS attack is a malicious attempt by a single person or a group of people to cause the victim, site, or node to deny service to its customers. When this attempt derives from a single host of the network, it constitutes a DoS attack. On the other hand, it is also possible that a lot of malicious hosts coordinate to flood the victim with an abundance of attack packets, so that the attack takes place simultaneously from multiple points. This type of attack is called a *Distributed DoS*, or DDoS attack.

DDoS Attack Description

DoS attacks attempt to exhaust the victim's resources. These resources can be network bandwidth, computing power, or operating system data structures. To launch a DDoS attack, malicious users first build a network of computers that they will use to produce the volume of traffic needed to deny services to computer users. To create this attack network, attackers discover vulnerable sites or hosts on the network. Vulnerable hosts are usually those that are either running no antivirus software or out-of-date antivirus software, or those that have not been properly patched. Vulnerable hosts are then exploited by attackers who use their vulnerability to gain access to these hosts. The next step for the intruder is to install new programs (known as *attack tools*) on the compromised hosts of the attack network. The hosts that are running these attack tools are known as *zombies*, and they can carry out any attack under the control of the attacker. Many zombies together form what we call an *army*.

But how can attackers discover the hosts that will make up the attack network, and how can they install the attack tools on these hosts? Though this preparation stage of the attack is very crucial, discovering vulnerable hosts and installing attack tools on them has become a very easy process. There is no need for the intruder to spend time in creating the attack tools because there are already prepared programs that automatically find vulnerable systems, break into these systems, and then install the necessary programs for the attack. After that, the systems that have been infected by the malicious code look for other vulnerable computers and install on them the same malicious code. Because of that widespread scanning to identify victim systems, it is possible that large attack networks can be built very quickly.

The result of this automated process is the creation of a DDoS attack network that consists of handler (master) and agent (slave, daemon) machines. It can be inferred from this process that another DDos attack takes place while the attack network is being built, because the process itself creates a significant amount of traffic.

Recruiting the Vulnerable Machines

Attackers can use different kinds of techniques (referred to as *scanning techniques*) in order to find vulnerable machines[1][2][3]. The most important follow:

- *Random scanning:* In this technique, the machine that is infected by the malicious code (such a machine can be either the attacker's machine or the machine of a member of their army, such as a zombie) probes IP addresses randomly from the IP address space and checks their vulnerability. When it finds a vulnerable machine, it breaks into it and tries to infect it, installing on it the same malicious code that is installed on itself. This technique creates significant traffic, because the random scanning causes a large number of compromised hosts to probe and check the same addresses. An advantage (to attackers) of this scanning method is that the malicious code can be spread very quickly because the scans seem to come from everywhere. However, the fast rate at which the malicious code is dispersed cannot last forever. After a small period of time, the spreading rate reduces because the number of the new IP addresses that can be discovered is smaller as time passes. This becomes obvious if we consider the analysis of David Moore and Colleen Shannon[4] on the spread of the Code-Red (CRv2) Worm, which uses random scanning to spread itself.
- *Hit-list scanning:* Long before attackers start scanning, they collect a list of a large number of potentially vulnerable machines. In their effort to create their army, they begin scanning down the list in order to find vulnerable machines. When they find one, they install on it the malicious code and divide the list in half. Then they give one half to the newly compromised machine, keep the other half, and continue scanning the remaining list. The newly infected host begins scanning down its list, trying to find a vulnerable machine. When it finds one, it implements the same procedure as described previously, and in this way the hit-list scanning takes place simultaneously from an enduringly increasing number of compromised machines. This mechanism ensures that the malicious code is installed on all vulnerable machines contained in the hit list in a short period of time. In addition, the hit list possessed by a new compromised host is constantly reducing because of the partitioning of the list discussed previously.

  As has been mentioned, the construction of the list is carried out long before the attackers start scanning. For that reason, the attackers can create the list at a very slow rate and for a long period of time. If the attackers conduct a slow scan, it is possible that this activity would not be noticed because a scanning process in a network usually occurs at extremely high frequencies, so a slow scan could occur without anyone realizing that it is a malicious scan.

It should also be mentioned that there are public servers such as the Netcraft Survey[2] that can create such hit lists without scanning.

- *Topological scanning:* Topological scanning uses information contained on the victim machine in order to find new targets. In this technique, an already-compromised host looks for URLs in the disk of a machine that it wants to infect. Then it renders these URLs targets and checks their vulnerability. The fact that these URLs are valid Web servers means that the compromised host scans possible targets directly from the beginning of the scanning phase. For that reason, the accuracy of this technique is extremely good, and its performance seems to be similar to that of hit-list scanning. Hence, topological scanning can create a large army of attackers extremely quickly and in that way can accelerate the propagation of the malicious code.
- *Local subnet scanning:* This type of scanning acts behind a firewall in an area that is considered to be infected by the malicious scanning program. The compromised host looks for targets in its own local network, using the information that is hidden in "local" addresses. More specifically, a single copy of the scanning program is running behind a firewall and tries to break into all vulnerable machines that would otherwise be protected by the firewall. This mechanism can be used in conjunction with other scanning mechanisms: for example, a compromised host can start its scans with local subnet scanning, looking for vulnerable machines in its local network. As soon as it has probed all local machines, it can continue the probing process by switching to another scanning mechanism in order to scan off-local network machines. In that way, an army with numerous zombies can be constructed at an extremely high speed.
- *Permutation scanning:* In this type of scanning, all machines share a common pseudorandom permutation list of IP addresses. Such a permutation list can be constructed using any block cipher of 32 bits with a preselected key[3]. If a compromised host has been infected during either the hit-list scanning or local subnet scanning, it starts scanning just after its point in the permutation list and scans through this list in order to find new targets. Otherwise, if it has been infected during permutation scanning, it starts scanning at a random point. Whenever it encounters an already-infected machine, it chooses a new random start point in the permutation list and proceeds from there. A compromised host can recognize an already-infected machine among noninfected ones, because such machines respond differently than other machines. The process of scanning stops when the compromised host encounters sequentially a predefined number of already-infected machines without finding new targets during that period of time. Then, a new permutation key is produced and a new scanning phase begins. This mechanism serves two major purposes: first, it prevents unnecessary reinfections of the same target because when a compromised host recognizes an already-compromised machine, it changes the way it scans according to the process described previously.

Second, this mechanism maintains the advantages (to attackers) of random scanning, because the scanning of new targets takes place in a random way. Hence, permutation scanning can be characterized as a coordinated scanning with an extremely good performance, because the randomization mechanism allows high scanning speeds.

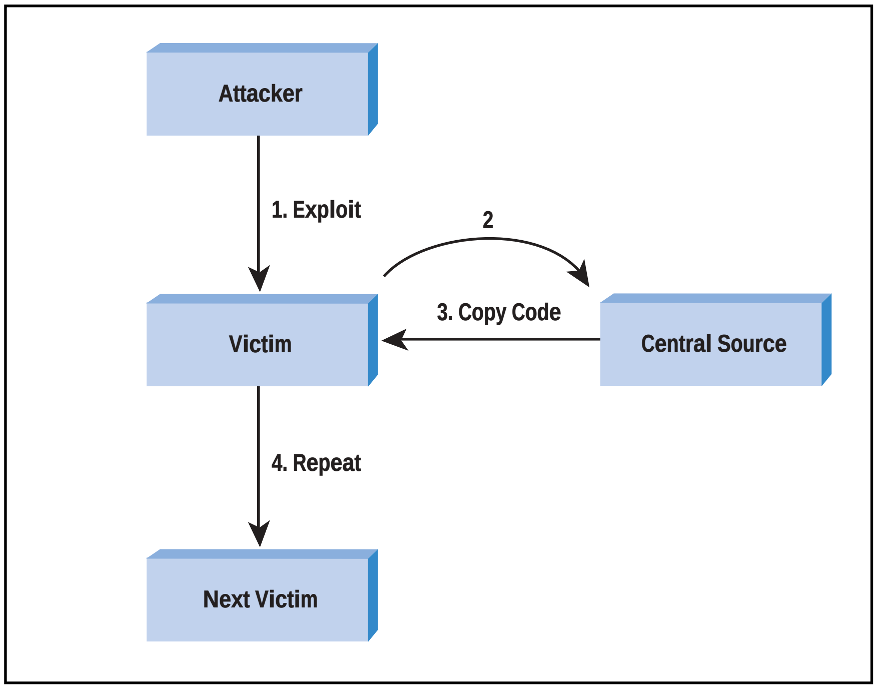
An improved version of permutation scanning is *partitioned permutation scanning*. This type of scanning is a combination of permutation and hit-list scanning. In this scenario, the compromised machine has a permutation list, which is cut in half when it finds a new target. Then it keeps one section of the list and gives the other section to the newly compromised machine. When the permutation list that an infected machine possesses reduces below a predefined level, the scanning scheme turns from partitioned permutation scanning into simple permutation scanning.

Propagating the Malicious Code

We can identify three groups of mechanisms for propagating malicious code and building attack networks[4]:

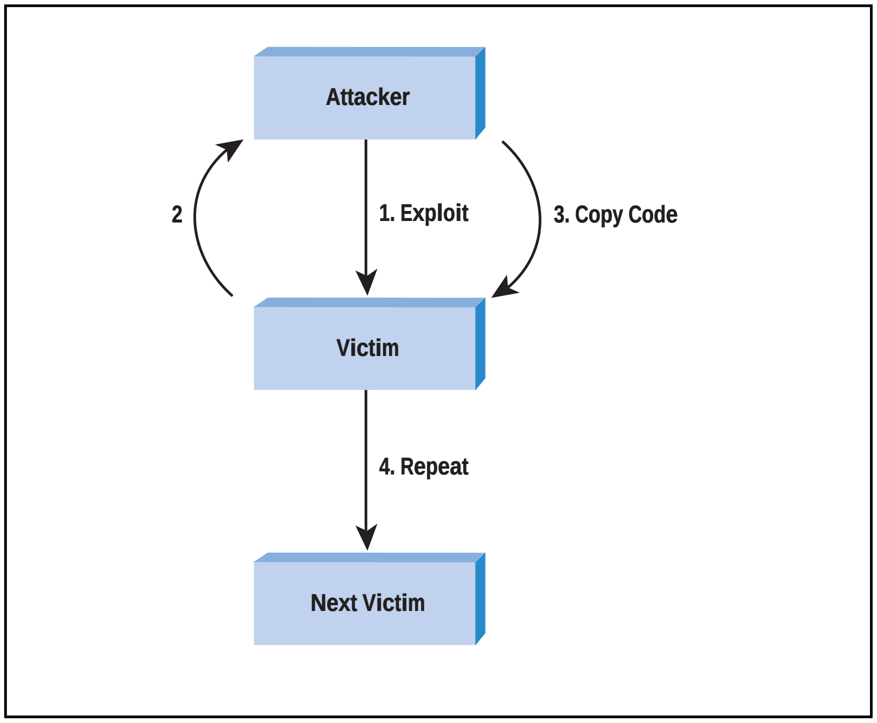
- *Central source propagation:* In this mechanism, after the discovery of the vulnerable system that will become one of the zombies, instructions are given to a central source so that a copy of the attack toolkit is transferred from a central location to the newly compromised system. After the toolkit is transferred, an automatic installation of the attack tools takes place on this system, controlled by a scripting mechanism. That initiates a new attack cycle, where the newly infected system looks for other vulnerable computers on which it can install the attack toolkit using the same process as the attacker. Like other file-transfer mechanisms, this mechanism commonly uses HTTP, FTP, and *remote-procedure call* (RPC) protocols. A graphical representation of this mechanism is shown in Figure 1.

Figure 1: Central Source Propagation



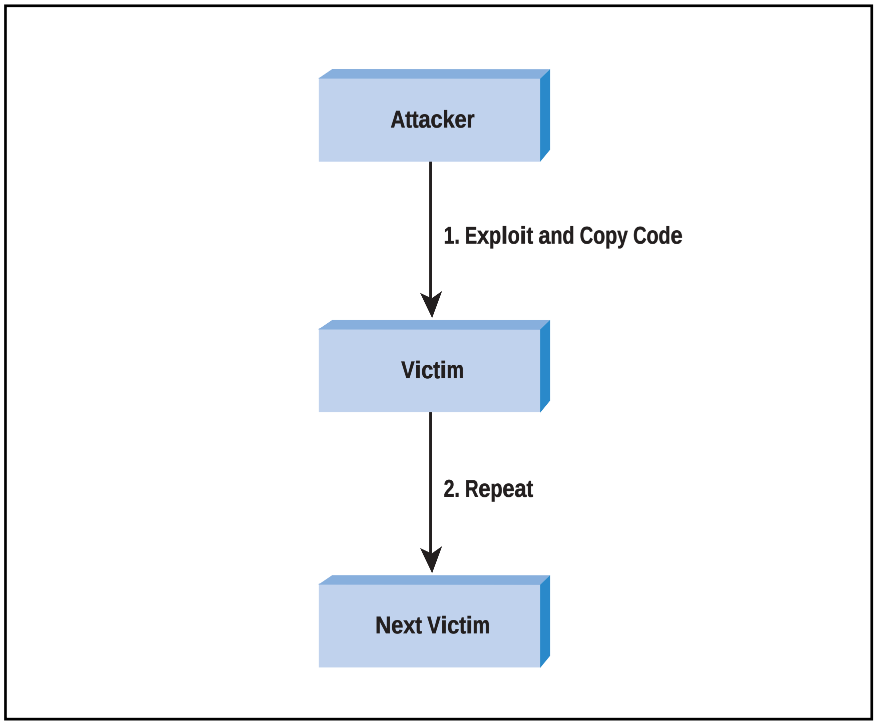
- *Back-chaining propagation:* In this mechanism, the attack toolkit is transferred to the newly compromised system from the attacker. More specifically, the attack tools that are installed on the attacker include special methods for accepting a connection from the compromised system and sending a file to it that contains the attack tools. This back-channel file copy can be supported by simple port listeners that copy file contents or by full intruder-installed Web servers, both of which use the *Trivial File Transfer Protocol* (TFTP). Figure 2 presents the this mechanism:

Figure 2: Back-Chaining Propagation



- *Autonomous propagation:* In this mechanism, the attacking host transfers the attack toolkit to the newly compromised system at the exact moment that it breaks into that system. This mechanism differs from the previously mentioned mechanisms in that the attack tools are planted into the compromised host by the attackers themselves and not by an external file source. Figure 3 shows the autonomous propagation.

Figure 3: Autonomous Propagation



After the construction of the attack network, the intruders use handler machines to specify the attack type and the victim's address and wait for the appropriate moment in order to mount the attack. Then, either they remotely command the launch of the chosen attack to agents or the daemons "wake up" simultaneously, as they had been programmed to do. The agent machines in turn begin to send a stream of packets to the victim, thereby flooding the victim's system with useless load and exhausting its resources. In this way, the attackers render the victim machine unavailable to legitimate clients and obtain unlimited access to it, so that they can inflict arbitrary damage. The volume of traffic may be so high that the networks that connect the attacking machines to the victim may also suffer from lower performance. Hence the provision of services over these networks is no longer possible, and in this way their clients are denied those services. Thus, the network that has been burdened by the attack load can be considered as one more victim of the DDos attack.

The whole procedure for carrying out a DDoS attack is mostly automated thanks to various attack tools. According to[5], the existence of the first controllable DDOS tool was reported by the *CERT Coordination Center* (CERT/CC) in early 1998 and it was called "Fapi." It is a tool that does not provide easy controls for setting up the DDoS network and does not handle networks with more than 10 hosts very well. In mid-1999 Trinoo arrived. Later that year the existence of *Tribe Flood Network* (TFN) and its upgraded version TFN2K (or TFN2000) was reported. Stacheldraht (German for "barbed wire") evolved out of the latter two tools (Trinoo and TFN). This tool is remarkable because it has full-control features and a Blowfish-encrypted control channel for the attacker. Moreover, in early 2000 it mutated into StacheldrahtV4, and later into Stacheldraht v1.666.

However, the development of attack tools did not stop, and many tools were later introduced, such as Mstream, Omega, Trinity, Derivatives, myServer, and Plague[6]. Dave Dittrich and his partners have provided the most comprehensive analyses of the Trinoo, Tribe Flood Network, Stacheldraht, shaft, and mstream DDoS attack tools[7]. Through this work, a lot of malicious code was captured, important observations were made about DDoS attack tools, and solutions were proposed toward detection and defense.

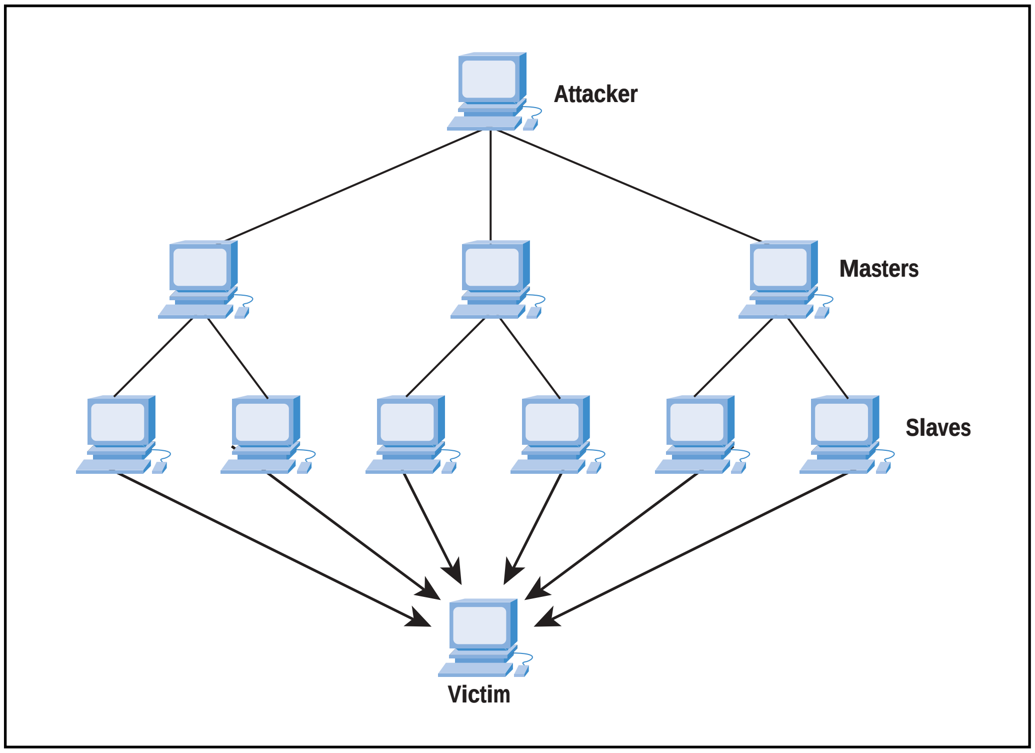
DDoS Attack Taxonomy

As has been already said, a DDoS attack takes place when many compromised machines infected by the malicious code act simultaneously and are coordinated under the control of a single attacker in order to break into the victim's system, exhaust its resources, and force it to deny service to its customers. There are mainly two kinds of DDoS attacks[10]: typical DDoS attacks and *distributed reflector DoS* (DRDoS) attacks. The following paragraphs describe these two kinds analytically.

Typical DDoS Attacks

In a typical DDoS attack, the army of the attacker consists of *master zombies* and *slave zombies*. The hosts of both categories are compromised machines that have arisen during the scanning process and are infected by malicious code. The attacker coordinates and orders master zombies and they, in turn, coordinate and trigger slave zombies. More specifically, the attacker sends an attack command to master zombies and activates all attack processes on those machines, which are in hibernation, waiting for the appropriate command to wake up and start attacking. Then, master zombies, through those processes, send attack commands to slave zombies, ordering them to mount a DDoS attack against the victim. In that way, the agent machines (slave zombies) begin to send a large volume of packets to the victim, flooding its system with useless load and exhausting its resources. Figure 4 shows this kind of DDoS attack.

Figure 4: A DDoS Attack



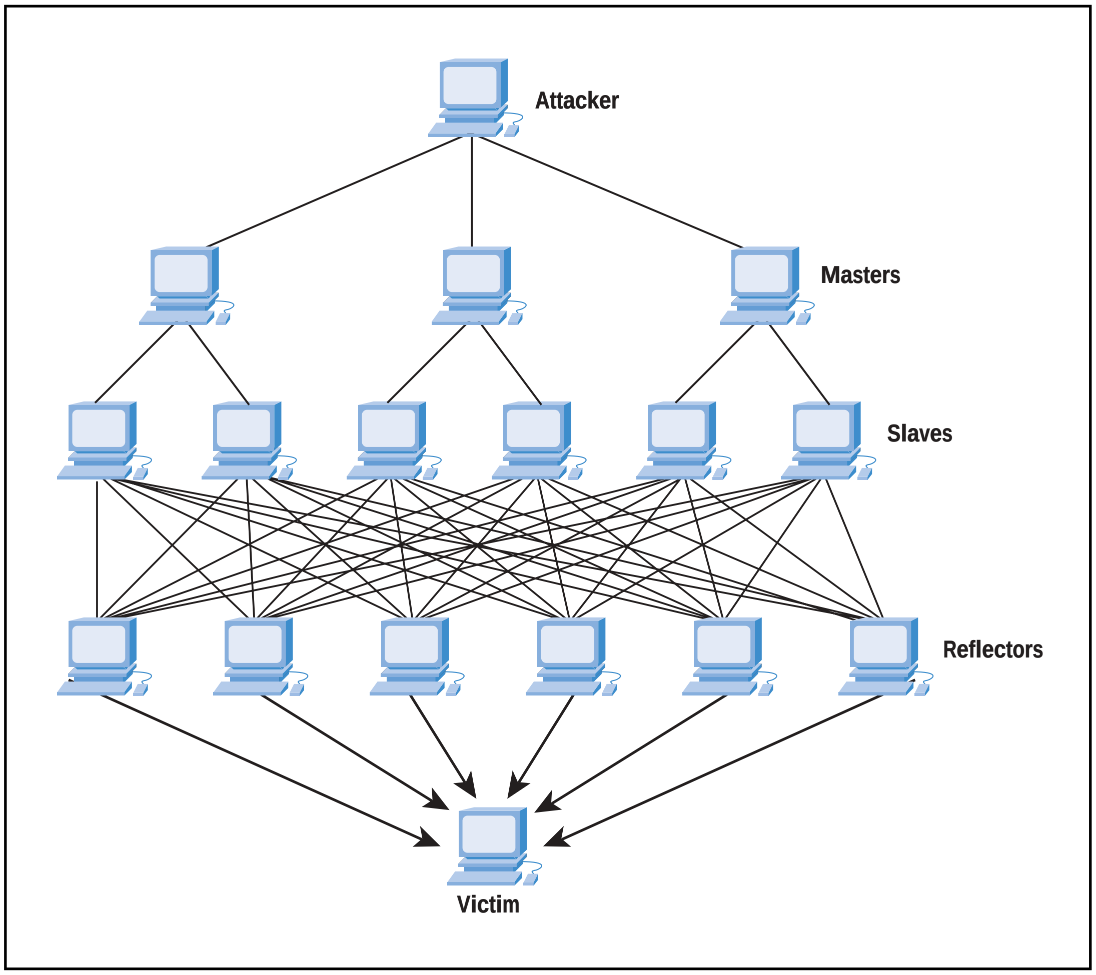
In cases of DDoS attacks, spoofed source IP addresses are used in the packets of the attack traffic. An attacker prefers to use such counterfeit source IP addresses for two major reasons: first, the attackers want to hide the identity of the zombies so that the victim cannot trace the attack back to them. The second reason concerns the performance of the attack. The attackers want to discourage any attempt of the victim to filter out the malicious traffic.

DRDoS Attacks

Unlike typical DDoS attacks, in DRDoS attacks the army of the attacker consists of master zombies, slave zombies, and reflectors[11]. The scenario of this type of attack is the same as that of typical DDoS attacks up to a specific stage. The attackers have control over master zombies, which, in turn, have control over slave zombies. The difference in this type of attack is that slave zombies are led by master zombies to send a stream of packets with the victim's IP address as the source IP address to other uninfected machines (known as *reflectors*), exhorting these machines to connect with the victim. Then the reflectors send the victim a greater volume of traffic, as a reply to its exhortation for the opening of a new connection, because they believe that the victim was the host that asked for it. Therefore, in DRDoS attacks, the attack is mounted by noncompromised machines, which mount the attack without being aware of the action.

Comparing the two scenarios of DDoS attacks, we should note that a DRDoS attack is more detrimental than a typical DDoS attack. This is because a DRDoS attack has more machines to share the attack, and hence the attack is more distributed. A second reason is that a DRDoS attack creates a greater volume of traffic because of its more distributed nature. Figure 5 graphically depicts a DRDoS attack.

Figure 5: A DRDoS Attack



Well-Known DDoS Attacks

This article would be incomplete without reference to some of the most well-known DDoS attacks. Some of the most famous documented DDoS attacks[12][13] are summarized in the following:

- *Apache2:* This attack is mounted against an Apache Web server where the client asks for a service by sending a request with many HTTP headers. However, when an Apache Web server receives many such requests, it cannot confront the load and it crashes.

- *ARP Poison: Address Resolution Protocol* (ARP) Poison attacks require the attacker to have access to the victim's LAN. The attacker deludes the hosts of a specific LAN by providing them with wrong MAC addresses for hosts with already-known IP addresses. This can be achieved by the attacker through the following process: The network is monitored for "arp who-has" requests. As soon as such a request is received, the malevolent attacker tries to respond as quickly as possible to the questioning host in order to mislead it for the requested address.
- *Back:* This attack is launched against an apache Web server, which is flooded with requests containing a large number of front-slash (/) characters in the URL description. As the server tries to process all these requests, it becomes unable to process other legitimate requests and hence it denies service to its customers.
- *CrashIIS:* The victim of a CrashIIS attack is commonly a Microsoft Windows NT IIS Web server. The attacker sends the victim a malformed GET request, which can crash the Web server.
- *DoSNuke:* In this kind of attack, the Microsoft Windows NT victim is inundated with "out-of-band" data (MSG_OOB). The packets being sent by the attacking machines are flagged "urg" because of the MSG_OOB flag. As a result, the target is weighed down, and the victim's machine could display a "blue screen of death."
- *Land:* In Land attacks, the attacker sends the victim a TCP SYN packet that contains the same IP address as the source and destination addresses. Such a packet completely locks the victim's system.
- *Mailbomb:* In a Mailbomb attack, the victim's mail queue is flooded by an abundance of messages, causing system failure.
- *SYN Flood:* A SYN flood attack occurs during the three-way handshake that marks the onset of a TCP connection. In the three-way handshake, a client requests a new connection by sending a TCP SYN packet to a server. After that, the server sends a SYN/ACK packet back to the client and places the connection request in a queue. Finally, the client acknowledges the SYN/ACK packet. If an attack occurs, however, the attacker sends an abundance of TCP SYN packets to the victim, obliging it both to open a lot of TCP connections and to respond to them. Then the attacker does not execute the third step of the three-way handshake that follows, rendering the victim unable to accept any new incoming connections, because its queue is full of half-open TCP connections.
- *Ping of Death:* In Ping of Death attacks, the attacker creates a packet that contains more than 65,536 bytes, which is the limit that the IP protocol defines. This packet can cause different kinds of damage to the machine that receives it, such as crashing and rebooting.

- *Process Table:* This attack exploits the feature of some network services to generate a new process each time a new TCP/IP connection is set up. The attacker tries to make as many uncompleted connections to the victim as possible in order to force the victim's system to generate an abundance of processes. Hence, because the number of processes that are running on the system cannot be boundlessly large, the attack renders the victim unable to serve any other request.
- *Smurf Attack:* In a "smurf" attack, the victim is flooded with *Internet Control Message Protocol* (ICMP) "echo-reply" packets. The attacker sends numerous ICMP "echo-request" packets to the broadcast address of many subnets. These packets contain the victim's address as the source IP address. Every machine that belongs to any of these subnets responds by sending ICMP "echo-reply" packets to the victim. Smurf attacks are very dangerous, because they are strongly distributed attacks.
- *SSH Process Table:* Like the Process Table attack, this attack makes hundreds of connections to the victim with the *Secure Shell* (SSH) Protocol without completing the login process. In this way, the daemon contacted by the SSH on the victim's system is obliged to start so many SSH processes that it is exhausted.
- *Syslogd:* The Syslogd attack crashes the *syslogd* program on a Solaris 2.5 server by sending it a message with an invalid source IP address.
- *TCP Reset:* In TCP Reset attacks, the network is monitored for "tcp-connection" requests to the victim. As soon as such a request is found, the malevolent attacker sends a spoofed TCP RESET packet to the victim and obliges it to terminate the TCP connection.
- *Teardrop:* While a packet is traveling from the source machine to the destination machine, it may be broken up into smaller fragments, through the process of fragmentation. A Teardrop attack creates a stream of IP fragments with their offset field overloaded. The destination host that tries to reassemble these malformed fragments eventually crashes or reboots.
- *UDP Storm:* In a *User Datagram Protocol* (UDP) connection, a character generation ("chargen") service generates a series of characters each time it receives a UDP packet, while an echo service echoes any character it receives. Exploiting these two services, the attacker sends a packet with the source spoofed to be that of the victim to another machine. Then, the echo service of the former machine echoes the data of that packet back to the victim's machine and the victim's machine, in turn, responds in the same way. Hence, a constant stream of useless load is created that burdens the network.

The first DoS attack occurred against Panix, the New York City area's oldest and largest *Internet Service Provider* (ISP), on September 6, 1996, at about 5:30 p.m.[14]. The attack was against different computers on the provider's network, including mail, news, and Web servers, user "login" machines, and name servers. The Panix attack was a SYN Flood attack deriving from random IP addresses and directed toward server *Simple Mail Transfer Protocol* (SMTP) ports. More specifically, Panix's computers were flooded by, on average, 150 SYN packets per second (50 per host), so Panix could not respond to legitimate requests[15]. Because the attackers used spoofed source IP addresses in their packets, the addresses could not be traced and malicious traffic could not be filtered. For that reason the attack was not immediately confronted. The solution was to use a special structure, instead of full *Transmission Control Block* (TCB), to hold half-open connections until the last ACK packet was received. In that way, the listen queue was large enough to keep all the SYN requests before the half-open connection timed out. The timeout, on the other hand, was adjusted to 94 seconds[16]. However, although Panix overcame this attack, the new threat (DoS attacks) made administrators worry.

Problems Caused and Countermeasures

The results of these attacks are disastrous. DDoS attacks have two characteristics: they are both distributed attacks and denial-of-service attacks. Distributed means that they are large-scale attacks having a great impact on the victims. Denial of service means that their goal is to deny the victim's access to a particular resource (service). This is not too difficult because the Internet was not designed with security in mind.

First, available *bandwidth* is one of the "goods" that attackers try to consume. Flooding the network with useless packets, for example, prevents legitimate ICMP echo packets from traveling over the network. Secondly, attackers try to consume *CPU power*. By generating several thousands of useless processes on the victim's system, attackers manage to fully occupy memory and process tables. In this way the victim's computer cannot execute any process and the system breaks down. Using this method, the attacker manages to prevent clients from accessing the victim's services and disrupts the current connections. Finally, attackers try to occupy victims' *services* so that no one else can access them. For example, by leaving TCP connections half open, attackers manage to consume the victim's data structures, and when they do so, no one else can establish a TCP connection with that victim.

The impact of these attacks is catastrophic, especially when victims are not individuals but companies. DDoS attacks prevent victims either from using the Internet, or from being reached by other people. Consequently, when the victim is an ISP, the results of such an attack are far more severe. ISPs' clients will not be served. E-business is also top on the "hit list." Being off line for a few hours could result in the loss of large sums of money for an ISP. Finally, the fact that companies use the Internet more and more for advertising or for providing goods and services increases the severity of such incidents.

Defense Mechanisms

From the beginning, all legitimate users have tried to respond against these threats. University communities and software corporations have proposed several methods against the DDoS threat. Despite the efforts, the solution remains a dream. The attackers manage to discover other weaknesses of the protocols and—what is worse—they exploit the defense mechanisms in order to develop attacks. They discover methods to overcome these mechanisms or they exploit them to generate false alarms and to cause catastrophic consequences.

Many experts have tried to classify the DDoS defense mechanisms in order to clarify them. This classification gives users an overall view of the situation and helps defense-mechanism developers cooperate against the threat. The basic discrimination is between *preventive* and *reactive* defense mechanisms.

Preventive Mechanisms

The preventive mechanisms try to eliminate the possibility of DDoS attacks altogether or to enable potential victims to endure the attack without denying services to legitimate clients. With regard to attack prevention, countermeasures can be taken on victims or on zombies. This means modification of the system configuration to eliminate the possibility of accepting a DDoS attack or participating unwillingly in a DDoS attack. Hosts should guard against illegitimate traffic from or toward the machine. By keeping protocols and software up-to-date, we can reduce the weaknesses of a computer. A regular scanning of the machine is also necessary in order to detect any “anomalous” behavior. Examples of system security mechanisms include monitoring access to the computer and applications, and installing security patches, firewall systems, virus scanners, and intrusion detection systems automatically. The modern trend is toward security companies that guard a client’s network and inform the client in case of attack detection to take defending measures. Several sensors monitor the network traffic and send information to a server in order to determine the “health” of the network. Securing the computer reduces the possibility of being not only a victim, but also a zombie. Not being a zombie is very important because it wipes out the attacker’s army. All these measures can never be 100-percent effective, but they certainly decrease the frequency and strength of DDoS attacks.

Many other measures can be taken in order to reduce the attacker’s army or restrict its “power.” Studying the attack methods can lead to recognizing loopholes in protocols. For example, administrators could adjust their network gateways in order to filter input and output traffic. The source IP address of output traffic should belong to the subnetwork, whereas the source IP address of input traffic should not. In this way, we can reduce traffic with spoofed IP addresses on the network[28].

Furthermore, over the last few years, several techniques have been proposed to test systems for possible drawbacks, before their shipment to the market. More precisely, by replacing the components of a system with malicious ones we can discover whether the system can survive an attack situation[38]. If the system breaks down, a drawback has been detected and developers must correct it.

On the other hand, DoS prevention mechanisms enable the victim to endure attack attempts without denying service to legitimate clients. Until now, two methods have been proposed for this scenario. The first one refers to policies that increase the privileges of users according to their behavior. When users' identities are verified, then no threat exists. Any illegitimate action from those users can lead to their legal prosecution. The second method is usually too expensive; it involves increasing the effective resources to such a degree that DDoS effects are limited. Most of the time application of such a measure is impossible.

Reactive Mechanisms

The reactive mechanisms (also referred to as *Early Warning Systems*) try to detect the attack and respond to it immediately. Hence, they restrict the impact of the attack on the victim. Again, there is the danger of characterizing a legitimate connection as an attack. For that reason it is necessary for researchers to be very careful.

The main detection strategies are *signature detection, anomaly detection,* and *hybrid systems.* Signature-based methods search for patterns (signatures) in observed network traffic that match known attack signatures from a database. The advantage of these methods is that they can easily and reliably detect known attacks, but they cannot recognize new attacks. Moreover, the signature database must always be kept up-to-date in order to retain the reliability of the system.

Anomaly-based methods compare the parameters of the observed network traffic with normal traffic. Hence it is possible for new attacks to be detected. However, in order to prevent a false alarm, the model of "normal traffic" must always be kept updated and the threshold of categorizing an anomaly must be properly adjusted.

Finally, hybrid systems combine both these methods. These systems update their signature database with attacks detected by anomaly detection. Again the danger is great because an attacker can fool the system by characterizing normal traffic as an attack. In that case an *Intrusion Detection System* (IDS) becomes an attack tool. Thus IDS designers must be very careful because their research can boomerang.

After detecting the attack, the reactive mechanisms respond to it. The relief of the impact of the attack is the primary concern. Some mechanisms react by limiting the accepted traffic rate. This means that legitimate traffic is also blocked. In this case the solution comes from traceback techniques that try to identify the attacker. If attackers are identified, despite their efforts to spoof their address, then it is easy to filter their traffic. Filtering is efficient only if attackers' detection is correct. In any other case filtering can become an attacker's tool.

The University of Washington provides an example of attack detection. Dave Dittrich and his team of 40 people discovered that more than 30 of their systems were zombies exploited by a single attacker[39]. By monitoring network traffic, Dittrich's team located directory and file names uncommon to the Windows operating systems the attacker ran on the network, as well as the port through which all these files were running communications.

Difficulties in Defending

Development of detection and defending tools is very complicated. Designers must think in advance of every possible situation because every weakness can be exploited. Difficulties involve:

- DDoS attacks flood victims with packets. This means that victims cannot contact anyone else in order to ask for help. So it is possible for a network neighbor to be attacked, but nobody would know it and nobody can help. Consequently, any action to react can be taken only if the attack is detected early. But can an attack be detected early? Usually traffic flow increases suddenly and without any warning[34][35][36]. For this reason defense mechanisms must react quickly.
- Any attempt of filtering the incoming flow means that legitimate traffic will also be rejected. And if legitimate traffic is rejected, how will applications that wait for information react? On the other hand, if zombies number in the thousands or millions, their traffic will flood the network and consume all the bandwidth. In that case filtering is useless because nothing can travel over the network.
- Attack packets usually have spoofed IP addresses. Hence it is more difficult to trace back to their source. Furthermore, it is possible that intermediate routers and ISPs may not cooperate in this attempt. Sometimes attackers, by spoofing source IP addresses, create counterfeit armies. Packets might derive from thousands of IP addresses, but zombies number only a few tens, for example.
- Defense mechanisms are applied in systems with differences in software and architecture. Also systems are managed by users with different levels of knowledge. Developers must design a platform independent of all these parameters.[37]

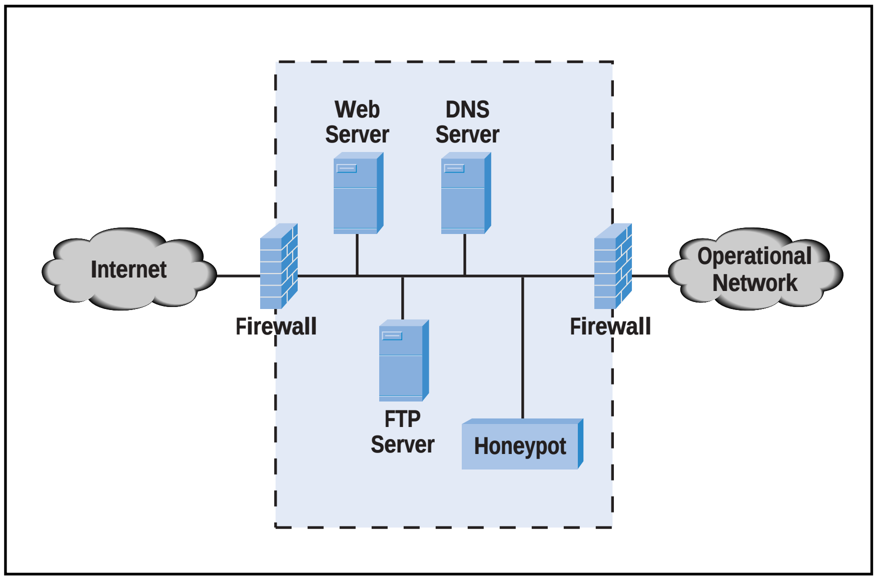
Modern Tendencies in Defending Against DDoS Attacks

Until now, developers have not managed to develop a 100-percent-effective defense mechanism. All mechanisms that have been presented either can confront only specific DDoS attacks or are being finally compromised by the attackers. Therefore, developers are currently working on DDoS diversion systems. *Honeypots* are the best representative of this category (See Figure 6).

Honeypots

There are two basic types of honeypots: *low-interaction honeypots* and *high-interaction honeypots*. The first ones refer to emulating services and operating systems. It is easy and safe to implement them. Attackers are not allowed to interact with the basic operating system, but only with specific services. For that reason, this type of honeypot cannot provide detailed informations for attackers' actions and they can easily be detected. However, they can detect communication attempts toward unused IP addresses. In that case an alarm is triggered, warning that someone is trying to compromise or attack the network. But what happens if the attack is not directed against the emulated service?

Figure 6: Honeypot



The answer comes from high-interaction honeypots. In [41], Honeynet is proposed. Honeynet is not a software solution that can be installed on a computer but a whole architecture, a network that is created to be attacked. Within this network, every activity is recorded and attackers are being trapped. Encrypted SSH sessions, e-mails, file uploads, and every possible attacker's action is captured. Moreover, a Honeywall gateway allows incoming traffic, but controls outgoing traffic using intrusion prevention technologies. This allows the attacker to interact with Honeynet systems, but prevents the attacker from harming other non-Honeynet systems. By studying the captured traffic, researchers can discover new methods and tools and they can fully understand attackers' tactics. However, Honeynet systems are more complex to install and deploy and the risk is increased as attackers interact with real operating systems and not with emulations. But what would happen if someone did compromise such a system? The consequences could be disastrous.

Route Filter Techniques

Different suggestions for defending against DDoS attacks derive from the *Border Gateway Protocol* (BGP) community. When routing protocols were designed, developers did not focus on security, but effective routing mechanisms and routing loop avoidance. Early on, attackers started directing their attention towards routers. By gaining access to a router, they could direct the traffic over bottlenecks, view critical data, and modify them. Cryptographic authentication mitigates these threats. Because of neighbor authentication, the routing update comes from a trusted source and there is no possibility that someone can give routers invalid routing information in order to compromise a network. On the other hand, routing filters are necessary for preventing critical routes and subnetworks from being advertised and suspicious routes from being incorporated in routing tables. In that way, attackers do not know the route toward critical servers and suspicious routes are not used.

Two other route filter techniques, *blackhole routing* and *sinkhole routing,* can be used when the network is under attack. These techniques try to temporarily mitigate the impact of the attack. The first one directs routing traffic to a null interface, where it is finally dropped. At first glance, it would be perfect to "blackhole" malicious traffic. But is it always possible to isolate malicious from legitimate traffic? If victims know the exact IP address being attacked, then they can ignore traffic originating from these sources. This way, the attack impact is restricted because the victims do not consume CPU time or memory as a consequence of the attack. Only network bandwidth is consumed. However, if the attackers' IP addresses cannot be distinguished and all traffic is blackholed, then legitimate traffic is dropped as well. In that case, this filter technique fails.

Sinkhole routing involves routing suspicious traffic to a valid IP address where it can be analyzed. There, traffic that is found to be malicious is rejected (routed to a null interface); otherwise it is routed to the next hop. A sniffer on the sinkhole router can capture traffic and analyze it. This technique is not as severe as the previous one. The effectiveness of each mechanism depends on the strength of the attack. Specifically, sinkholing cannot react to a severe attack as effectively as blackholing. However, it is a more sophisticated technique, because it is more selective in rejecting traffic.

Filtering malicious traffic seems to be an effective countermeasure against DDoS. The closer to the attacker the filtering is applied, the more effective it is. This is natural, because when traffic is filtered by victims, they "survive," but the ISP's network is already flooded. Consequently, the best solution would be to filter traffic on the source; in other words, filter zombies' traffic.

Until now, three filtering possibilities have been reported concerning criteria for filters. The first one is filtering on the *source address*. This one would be the best filtering method, if we knew each time who the attacker is. However, this is not always possible because attackers usually use spoofed IP addresses. Moreover, DDoS attacks usually derive from thousands of zombies and this makes it too difficult to discover all the IP addresses that carry out the attack. And even if all these IP addresses are discovered, the implementation of a filter that rejects thousands of IP addresses is practically impossible to deploy.

The second filtering possibility is filtering on the *service*. This tactic presupposes that we know the attack mechanism. In this case, we can filter traffic toward a specific UDP port or a TCP connection or ICMP messages. But what if the attack is directed toward a very common port or service? Then we must either reject every packet (even if it is legitimate) or suffer the attack.

Finally, there is the possibility of filtering on the *destination address*. DDoS attacks are usually addressed to a restricted number of victims, so it seems to be easy to reject all traffic toward them. But this means that legitimate traffic is also rejected. In case of a large-scale attack, this should not be a problem because the victims will soon break down and the ISP will not be able to serve anyone. So filtering prevents victims from breaking down by simply keeping them isolated.

Fred Baker and Paul Ferguson developed an technique called *Ingress Filtering* for mitigating DoS attacks (and, later, DDoS attacks too). After the Panix attack and a few other attacks, Paul Ferguson wrote RFC 2267[42], which became *Best Current Practices* (BCP) 38 in RFC 2827[43]. This RFC presents a method for using ingress traffic filtering against DoS attacks that use forged IP addresses and try to be propagated from "behind" an ISP's aggregation point. This method prevents the attack from forged source addresses, but nothing can be done against an attack from a valid source address. However, in that case, if the attack is detected, it is easy to trace the attacker. Finally, although this solution allows the network to protect itself from other attacks too (for example, spoofed management access to networking equipment), it can also create some problems, for example, with multihoming.

For that reason, RFC 2827 was recently (March 2004) updated by Fred Baker in BCP 84/ RFC 3704[44]. This RFC describes and evaluates the current ingress filtering mechanisms, examines some implementation matters related to ingress filtering, and presents some solutions to ingress filtering with multihoming. According to this RFC, ingress filtering should be implemented at multiple levels in order to prohibit the use of spoofed addresses and to make attackers more traceable, even if asymmetric/multihomed networks are presented. However, although Ferguson's work was published a long time ago, service providers in some cases ignore his suggestions.

Hybrid Methods and Guidelines

Currently researchers try to combine the advantages from all the methods stated previously in order to minimize their disadvantages. As a result, several mechanisms that implement two or more of these techniques are proposed for mitigation of the impact of DDoS attacks. The best solution to the DDoS problem seems to be the following: victims must detect that they are under attack as early as possible. Then they must trace back the IP addresses that caused the attack and warn zombies administrators about their actions. In that way, the attack can be confronted effectively.

However, as we saw previously, this is currently impossible. The lack of a 100-percent-effective defending tool imposes the necessity of private alerts. Users must care for their own security. Some basic suggestions follow:

- Prevent installation of distributed attack tools on our systems. This will help to restrict the zombies army. Several tasks also need to be performed. First, keep protocols and operating systems up-to-date. We can prevent system exploitation by eliminating the number of weaknesses of our system.
- Use firewalls in gateways to filter incoming and outgoing traffic. Incoming packets with source IP addresses belonging to the subnetwork and outgoing packets with source IP addresses not belonging to the subnetwork are not logical.
- Deploy IDS systems to detect patterns of attacks.
- Deploy antivirus programs to scan malicious code in our system.

Further Thoughts

The Internet is not stable—it reforms itself rapidly. This means that DDoS countermeasures quickly become obsolete. New services are offered through the Internet, and new attacks are deployed to prevent clients from accessing these services. However, the basic issue is whether DDoS attacks represent a network problem or an individual problem—or both. If attacks are mainly a network problem, a solution could derive from alterations in Internet protocols. Specifically, routers could filter malicious traffic, attackers could not spoof IP addresses, and there would be no drawback in routing protocols. If attacks are mostly the result of individual system weaknesses, the solution could derive from an effective IDS system, from an antivirus, or from an invulnerable firewall. Attackers then could not compromise systems in order to create a "zombies" army. Obviously, it appears that both network and individual hosts constitute the problem. Consequently, countermeasures should be taken from both sides. Because attackers cooperate in order to build the perfect attack methods, legitimate users and security developers should also cooperate against the threat. The solution will arise from combining both network and individual countermeasures.

References

[1] Kevin Tsui, "Tutorial-Virus (Malicious Agents)," University of Calgary, October 2001.

[2] Nicholas Weaver, "Warhol Worms: The Potential for Very Fast Internet Plagues," `http://www.iwar.org.uk/comsec/resources/worms/warhol-worm.htm`

[3] Nicholas Weaver, U.C. Berkeley BRASS group, "Potential Strategies for High Speed Active Worms: A Worst Case Analysis," February 2002

[4] David Moore and Colleen Shannon, "The Spread of the Code Red Worm (crv2)," July 2001, `http://www.caida.org/analysis/security/codered/coderedv2_analysis.xml#animations`

[5] "A Chronology of CERT Coordination Center Involvement with Distributed Denial-of-Service Tools," `http://www.cdt.org/security/dos/000229senatehouse/chron.html`

[6] "Analyzing Distributed Denial Of Service Tools: The Shaft Case," Sven Dietrich, NASA Goddard Space Flight Center; Neil Long, Oxford University; David Dittrich, University of Washington, `http://www.usenix.org/events/lisa2000/full_papers/dietrich/dietrich_html/`

[7] `http://staff.washington.edu/dittrich`

[8] Kevin J. Houle, CERT/CC; George M. Weaver, CERT/CC, in collaboration with: Neil Long, Rob Thomas, "Trends in Denial of Service Attack Technology," V1.0, October 2001.

[9] `http://staff.washington.edu/dittrich/misc/stacheldraht.analysis`

[10] T. Peng, C. Leckie, and K. Ramamohanarao, "Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring," The University of Melbourne, Australia, 2003.

[11] Steve Gibson, "Distributed Reflection Denial of Service Description and Analysis of a Potent, Increasingly Prevalent, and Worrisome Internet Attack," February 2002.

[12] `http://www.ll.mit.edu/IST/ideval/docs/1999/attackDB.html`

[13] Yanet Manzano, "Tracing the Development of Denial of Service Attacks: A Corporate Analogy," 2003, `http://www.acm.org/crossroads/xrds10-1/tracingDOS.html`

[14] `http://www.panix.com/press/synattack.html`

[15] `http://cypherpunks.venona.com/date/1996/09/msg01055.html`

[16] `http://cypherpunks.venona.com/date/1996/09/msg01061.html`

[17] Larry Rogers, “What Is a Distributed Denial of Service (DDoS) Attack and What Can I Do About It?” February 2004, **`http://www.cert.org/homeusers/ddos.html`**

[18] Alefiya Hussain, John Heidemann, and Christos Papadopoulo, “A Framework for Classifying Denial of Service Attacks,” 25 February 2003.

[19] **`http://www.cs.berkeley.edu/~nweaver/warhol.old.html`**

[20] CIS 659 “Introduction to Network Security – Fall 2003,” **`http://www.cis.udel.edu/~sunshine/F03/CIS659/class15.pdf`**

[21] Miguel Vargas Martin, School of Computer Science, Carleton University, “Overview of Worms and Defence Strategies,” October 2003.

[22] “Computer Security,” Testimony of Richard D. Pethia, Director, CERT Centers Software Engineering Institute, Carnegie Mellon University, March 2000, **`http://www.cert.org/congressional_testimony/Pethia_testimony_Mar9.html#Distributed`**

[23] Jelena Mirkovic, Janice Martin, and Peter Reiher, UCLA, “A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms.”

[24] Distributed Denial of Service Tools, **`http://www.cert.org/incident_notes/IN-99-07.html`**

[25] Barbara Fraser, Lawrence Rogers, and Linda Pesante, “Was the Melissa Virus So Different?” *The Internet Protocol Journal*, Volume 2, No. 2, June 1999.

[26] **`http://news.bbc.co.uk/1/hi/sci/tech/635444.stm`**

[27] **`http://www.nta-monitor.com/newrisks/feb2000/yahoo.htm`**

[28] **`http://www.cert.org/advisories/CA-1996-21.html`**

[29] S. Axelsson, “Intrusion Detection Systems: A Survey and Taxonomy,” Technical Report 99-15, Department of Computer Engineering, Chalmers University, March 2000.

[30] J. Shapiro and N. Hardy, “EROS: A principle-driven Operating System from the Ground Up,” *IEEE Software*, pp. 26–33, January/February 2002.

[31] A. Garg and A. L. Narasimha Reddy, “Mitigating Denial of Service Attacks Using QoS Regulation,” Texas A & M University Tech report, TAMU-ECE-2001-06.

[32] Y. L. Zheng and J. Leiwo, “A method to implement a Denial of Service Protection Base,” *Information Security and Privacy*, Volume 1270 of *Lecture Notes in Computer Science* (LNCS), pp. 90–101, 1997.

[33] CERT on Home Network Security:
`http://www.cert.org/tech_tips/home_networks.html`

[34] CERT on SMURF Attacks:
`http://www.cert.org/advisories/CA-1998-01.html`

[35] CERT on TCP SYN Flooding Attacks:
`http://www.cert.org/advisories/CA-1996-21.html`

[36] CERT TRIN00 Report:
`http://www.cert.org/incident_notes/IN-99-07.html#trinoo`

[37] `http://falcon.jmu.edu/~flynngn/whatnext.htm`

[38] Charalampos Patrikakis, Thomas Kalamaris, Vaios Kakavas, "Performing Integrated System Tests Using Malicious Component Insertion," *Electronic Notes in Theoretical Computer Science,* Volume 82 No. 6 (2003).

[39] `http://www.paypal.com/html/computerworld-011402.html`

[40] Ho Chung, "An Evaluation on Defensive Measures against Denial-of-Service Attacks," Fall 2002.

[41] Nathalie Weiler, "Honeypots for Distributed Denial of Service Attacks,"
`www.tik.ee.ethz.ch/~weiler/papers/wetice02.pdf`

[42] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2267, January 1998.

[43] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827, May 2000.

[44] F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks," RFC 3704, March 2004.

[45] Taxonomies of Distributed Denial of Service Networks, Attacks, Tools, and Countermeasures:
`www.ee.princeton.edu/~rblee/DDoS%20Survey%20Paper_v7final.doc`

[46] Lance Spitzner, "Honeypots Definitions and Value of Honeypots," May 2003, `http://www.tracking-hackers.com`

[47] How to Get Rid of Denial of Service Attacks:
`http://www.bgpexpert.com/antidos.php`

[48] Proposed Solutions to DDoS Information, March 2001:
`http://www.cs.virginia.edu/~survive/ddos/ddos_solutions.html`

[49] Dennis Fisher, "Thwarting the Zombies," March 2003:
`http://www.eweek.com/article2/0,3959,985389,00.asp`

[50] Merike Kaeo, "Route to Security," March 2004,
http://infosecuritymag.techtarget.com/ss/0,295796,sid6_iss346_art668,00.html

[51] "Report to the President's Commission on Critical Infrastructure Protection," James Ellis, David Fisher, Thomas Longstaff, Linda Pesante, and Richard Pethia, January 1997,
http://www.cert.org/pres_comm/cert.rpcci.body.html

[52] "Cisco Quality of Service and DDOS, Engineering Issues for Adaptive Defense Network," MITRE, 7/25/2001.

[53] "Denial of Service Attacks," CERT Coordination Center, June 4, 2001,
http://www.cert.org/tech_tips/denial_of_service.html

[54] Tom Chen, "Trends in Viruses and Worms," *The Internet Protocol Journal,* Volume 6, No. 3, September 2003.

CHARALAMPOS Z. PATRIKAKIS holds a Dipl.-Ing. and a Ph.D. degree from the Electrical Engineering and Computer Science Department of the National Technical University of Athens (NTUA). He is currently a senior research associate of the Telecommunications Labor-atory of NTUA. He has participated in several European Union projects (ESPRIT, RACE, ACTS, IST). His main interests are in the area of IP service design and implementation, multicasting in IP networks, IP transport protocols, and media streaming over IP networks. He is a member of IEEE, a member of the Greek Computer Society, a certified trainer by the National Accreditation Centre of Vocational Training Structures and Accompanying Support Services, and a member of the Technical Chamber of Greece. He can be reached at: **bpatr@telecom.ntua.gr**

MICHALIS MASIKOS holds a Dipl.-Ing. degree from the Electrical Engineering and Computer Science Department of the National Technical University of Athens (NTUA). He is currently a research associate of the Telecommunications Laboratory of NTUA. His interests are in the fields of network security, network simulation, and analysis. He can be reached at: **mmasik@telecom.ntua.gr**

OLGA ZOURARAKI holds a Dipl.-Ing. degree from the Electrical Engineering and Computer Science Department of the National Technical University of Athens (NTUA). She is currently a research associate of the Telecommunications Laboratory of NTUA. Her interests are in the fields of network security, Internet application design, and implementation. She can be reached at: **ozour@telecom.ntua.gr**

Letter to the Editor

Ole,

I was reading your latest issue of IPJ (Volume 7, No. 3, September 2004) and I could be wrong but I think you mis-typed an explanation about the STUN protocol. On page 12, 3rd paragraph, last sentence, it reads: "A received response indicates the presence of a port-restricted cone, and the lack of a response indicates the presence of a restricted cone."

According to the definitions you gave about "restricted cone" and "port-restricted cone" on pages 10 and 11. Shouldn't this sentence instead read: "A received response indicates the presence of a restricted cone, and the lack of a response indicates the presence of a port-restricted cone."

—*Ryan Liles*
`ryanliles@hotmail.com`

The author responds:

Ryan is correct, there is an error here in the text.

The flow control of the sequence of STUN tests is detailed in Figure 9 of the article. The test referred to here is to determine if the NAT is a restricted cone NAT, or a port-restricted cone NAT.

The restricted cone NAT, in Figure 7, is one where the NAT binding is accessible using any source port number on the external host when responding to a UDP packet from the internal sending host.

The port-restricted cone NAT, in Figure 8, is one where the NAT binding is accessible using the same port number as originally used by the internal lost host, and this binding is accessible from any external IP address.

The test referenced in this section, as per Figure 9, is one where the local host requests the external agent to respond using the same port number, but an altered source address. The text should read "This fourth request includes a control flag to direct the STUN server to respond using the alternate IP address, but with the same port value," in which case the interpretation of the response—that a response indicates the presence of a port-restricted cone NAT and the lack of response indicates the presence of a restricted cone NAT—would be correct.

Ryan is also correct in that if the test is performed the other way, requesting the agent to use the same IP address, but with the alternate port value, then the opposite interpretation would hold, namely that a response indicates the presence of a restricted cone NAT, and the lack of a response would indicate the presence of a port-restricted cone NAT, as Ryan points out.

Thanks to Ryan for following through this rather complex explanation of the STUN algorithm and spotting this error.

Regards,

—*Geoff Huston, APNIC*
`gih@apnic.net`

Book Review

The IP Multimedia Subsystem

The IP Multimedia Subsystem—Merging the Internet and the Cellular Worlds, by Gonzalo Camarillo and Miguel A. Garcia-Martin, John Wiley & Sons, 2004. ISBN 0470 87156 3.

The Internet and the cellular telephony system are the two most influential communication systems of the last half century. That the telecommunications industry would attempt to merge them into a single system was inevitable. The potential benefits are compelling—a single packet-based communication system with the capability to carry voice, video and data while providing ubiquitous wireless access and global mobility. The resulting system architecture is called the *Internet Multimedia Subsystem* (IMS) and is described comprehensively in this volume by Gonzalo Camarillo and Miguel A. Garcia Martin.

A "merging" of the two systems is only superficially what has happened. In practice, the IMS is an "embrace and extend" exercise which adapts the IP protocol suite to the existing architecture of the cellular telephony system. The cellular industry has taken a broad collection of IP protocols and mapped them onto their existing architecture, effecting a "protocol transplant" into an environment somewhat different from the Internet. Among the protocols imported are IPv6, SIP, DHCP, DNS, SDP, RTP, IPSec, and DIAMETER. Many are adopted unaltered; some are profiled by introducing new configuration data and rules; others are extended in various ways. The authors navigate their way through the various parts of the system with clarity and confidence. They can speak with authority on the subject—both were major contributors to the design through their key roles in the IETF and 3GPP (*Third Generation Partnership Project*—the standardization body for third generation cellular systems).

The book is clearly written and logically organized. The first part explains the reasoning behind adopting Internet-style packet networking for cellular mobile systems and describes the evolution of the standardization efforts. Although interesting, much of this material can be skimmed by those only interested in the meaty technical material which follows. The authors then explain the general principles behind the IMS architecture, including how various requirements of the cellular telephony industry drove the choices, and particularly the perceived need to extend and adapt the protocols rather than use them as deployed on the Internet. The majority of the book is devoted to ex-plaining in considerable technical depth how the protocols have been modified and how they are intended to work when IMS is successfully deployed. While not for the faint of heart, the writing is extremely clear and logical and hence should be understandable by anyone with a moderate background in the principles of protocol and system design. One aspect of the organization is particularly helpful to readers unfamiliar with some of the protocols in their native Internet instantiation. The authors divide the material into blocks where they first describe the native Internet flavor of the protocol, and then introduce the IMS-specific extensions and modifications.

Much of the volume is devoted to the *Session Initiation Protocol* (SIP) as the core signaling plane for IMS. All aspects of session establishment and management are covered. In addition, the ancillary parts of the control system are covered, including *Authentication, Authorization, and Accounting* (AAA), Security, Session Policies, and Quality of Service. For completeness, the data plane is also covered briefly through a discussion of the 3GPP audio, video, and text encoders, plus material on the media transport protocols.

The book concludes with a substantial section on how services are build on top of the core IMS protocols. Two of the most important, *Presence* and *Instant Messaging*, get comprehensive treatment, with a briefer discussion of the push-to-talk application.

As an old time "IP-head," it is hard to come away from this deep exploration of IMS without a bit of trepidation. The hallmark of IP and the Internet are simplicity and generality. IMS arguably succeeds at the latter, but at the expense of almost numbing complexity. This was perhaps inevitable given that the goal was to adapt Internet packet technology to the cellular system, which is itself quite complex. IMS will be quite a challenge to deploy. It remains to be seen if transplanting IP into a cellular telephony architectural model will result in economically sustainable services for the service providers or if a more native peer-to-peer Internet approach will simply bypass all the fancy IMS elements and just use basic packet transport. Such a market experiment is currently playing out in the broadband access arena with the broadband pipe suppliers offering telephony-oriented services themselves via customized standards like PacketCable, while third parties like Vonage and Skype simply piggyback on basic IP packet transport.

The next few years will be interesting. Whatever the outcome, anyone needing to be technically conversant with the architecture and protocols of IMS will find *The IP Multimedia Subsystem* indispensable.

—David Oran
`oran@cisco.com`

Read Any Good Books Lately?

Then why not share your thoughts with the readers of IPJ? We accept reviews of new titles, as well as some of the "networking classics." In some cases, we may be able to get a publisher to send you a book for review if you don't have access to it. Contact us at **`ipj@cisco.com`** for more information.

Call for Papers

The Internet Protocol Journal (IPJ) is published quarterly by Cisco Systems. The journal is not intended to promote any specific products or services, but rather is intended to serve as an informational and educational resource for engineering professionals involved in the design, development, and operation of public and private internets and intranets. The journal carries tutorial articles ("What is…?"), as well as implementation/operation articles ("How to…"). It provides readers with technology and standardization updates for all levels of the protocol stack and serves as a forum for discussion of all aspects of internetworking.

Topics include, but are not limited to:

- Access and infrastructure technologies such as: ISDN, Gigabit Ethernet, SONET, ATM, xDSL, cable, fiber optics, satellite, wireless, and dial systems
- Transport and interconnection functions such as: switching, routing, tunneling, protocol transition, multicast, and performance
- Network management, administration, and security issues, including: authentication, privacy, encryption, monitoring, firewalls, trouble-shooting, and mapping
- Value-added systems and services such as: Virtual Private Networks, resource location, caching, client/server systems, distributed systems, network computing, and Quality of Service
- Application and end-user issues such as: e-mail, Web authoring, server technologies and systems, electronic commerce, and application management
- Legal, policy, and regulatory topics such as: copyright, content control, content liability, settlement charges, "modem tax," and trademark disputes in the context of internetworking

In addition to feature-length articles, IPJ will contain standardization updates, overviews of leading and bleeding-edge technologies, book reviews, announcements, opinion columns, and letters to the Editor.

Cisco will pay a stipend of US$1000 for published, feature-length articles. Author guidelines are available from Ole Jacobsen, the Editor and Publisher of IPJ, reachable via e-mail at **ole@cisco.com**

The Internet Protocol Journal is published quarterly by the Chief Technology Office, Cisco Systems, Inc.
www.cisco.com
Tel: +1 408 526-4000
E-mail: ipj@cisco.com

CISCO SYSTEMS

The Internet Protocol Journal, Cisco Systems
170 West Tasman Drive, M/S SJ-7/3
San Jose, CA 95134-1706
USA

ADDRESS SERVICE REQUESTED

PRSRT STD
U.S. Postage
PAID
PERMIT No. 5187
SAN JOSE, CA