IEEE COMMUNICATIONS SURVEYS
The Electronic Magazine of Original Peer-Reviewed Survey Articles

BUILDING AN INTEGRATED SECURITY GATEWAY: MECHANISMS, PERFORMANCE EVALUATIONS, IMPLEMENTATIONS, AND RESEARCH ISSUES

YING-DAR LIN, HUAN-YUN WEI, AND SHAO-TANG YU, NATION CHIAO TUNG UNIVERSITY

ABSTRACT

Network security has become a critical issue for enterprises. This article first gives a tutorial of each basic component of a security gateway, including the firewall, content filtering, network address translation (NAT), the virtual private network (VPN), and the intrusion detection system (IDS). The building of an integrated security gateway, using various open-source packages, is then described. Conflicts among the packages are resolved to ensure interoperability. Next, we internally/externally evaluate the performance of each component with six commercial implementations to identify the problems for future research directions. Readers can understand how these components deliver secure operations, how a packet can properly traverse through such a gateway, and how many resources are consumed in each software component. Selected packages include the Linux kernel, ipchains (packet filter), Squid (URL filter), FWTK (content filter), FreeS/WAN (VPN), and Snort (IDS). ipchains and FreeS/WAN are found viable, but FWTK and Snort suffer performance problems. Further examining their source code and data structures reveals the improper implementation in FWTK and the less scalable linear matching algorithms in ipchains and Snort. Finally, several approaches to scale up these software components are suggested to improve the performance. Note that installing such a security gateway does not mean secured. This study focuses on building a product-like security gateway and on evaluating its performance. The integrated system with a self-developed Web management console is publicly available for downloading.

The Internet has moved from a convenience to a mission-critical platform for conducting and succeeding in business. As Internet applications prevail, security problems due to intruders arise. To guarantee the secure operations of an enterprise network, access control to block unauthorized inbound/outbound traffic, encryption/authentication to protect traffic from interception/modification/fabrication, and intrusion detection to discover attacks, must be in place at the edge of the enterprise network. Firewalls, virtual private networks (VPN), and intrusion detection systems (IDS), respectively, address the above three requirements. Originally they were separate devices, but they have been integrated with other services such as network address translation (NAT) as a single security gateway. Their functions, together with NAT, are described in the following sections.

FIREWALL

Firewalls can provide basic or advanced access control over pass-through connections and are classified into three categories

- Packet filtering firewall (PF)
- Stateful inspection packet filtering firewall (SPF)
- Application proxy firewall (APF)

A PF filters individual packets according to the access control rules, which are specified using the *condition* and *action* fields. Condition defines the packet-matching criteria, such as a certain subnet or an application, by investigating header fields such as protocol identifier, source/destination MAC/IP/port, and various TCP/IP flags. However, a PF cannot inspect the *correlation* among consecutive packets. An SPF records relevant information of a connection to trace the validity of each

packet in this connection. Thus, an SPF can block invalid packets according to standard protocol specifications, such as TCP state transitions, TCP window size/Acknowledgments, and IP fragmentation bits. Moreover, SPF simplifies management. For example, if the access control rules deny all outgoing/incoming connections except those outgoing to the external Web servers, an SPF can automatically open a door for the feedback Web contents. For advanced access control to filter the contents of a connection, such as denying outgoing HTTP requests to specific URLs (URL filtering), or blocking incoming Java/ActiveX/cookie Web objects, reassembling packets is required to investigate application-layer headers or payloads. An APF works as a user-space process at the application layer and lets the underlying kernel TCP/IP stack reassemble the packets. It acts as a "middleman" to forward/check/intercept the content of the requests or responses for the application. So an APF is also known as a content filter.

Virtual Private Network (VPN)

A virtual private network can establish secured virtual links among different organizations, such as branch offices. Tunneling [1] by appending additional headers facilitates the virtual lease line while cryptographic technologies prevent private information passing through the public Internet from being intercepted, modified, or fabricated. However, when complex cryptographic algorithms are employed, encryption and decryption within VPN tunnels becomes the performance bottleneck. Hence, dedicated hardware has been proposed to maximize the throughput and minimize the latency. Modern VPN technologies include PPTP [2], L2TP [3], and IPsec [4]. PPTP and L2TP work at the data link layer and are suitable for secure remote access between mobile users and enterprises. In contrast, IPsec works at the network layer and can provide secured tunnels among subnets. IPsec provides encryption and authentication mechanisms for the IP protocol suite. Encryption prevents intruders from reading information by sniffing traffic among hosts. Authentication prevents intruders from spoofing the hosts of a connection. Nowadays, IPsec has become a must for the VPN service in a security gateway. Accordingly, this study focuses on IPsec.

Intrusion Detection System (IDS)

Many network intrusions cannot be identified until the traffic has been passively analyzed. For example, denial of service (DoS) attacks such as ICMP-flooding are difficult to recognize until numerous ICMP packets have arrived within a small time interval; application-specific buffer-overflow[1] attacks to obtain root privilege, such as subverting an FTP server by a long "MKDIR" command, may require buffering and reassembling several packets before seeing the whole FTP command. A network-based IDS can detect such attacks by matching a substring, for example, the "phf" in "GET/cgi-bin/phf?," to identify those network packets as vehicles of a Web server attack. When such kinds of potential hostile activities are detected, an IDS will alert system administrators and may block the activity.

The above examples describe the basic functions of a network-based IDS. In fact, the IDS model can be host-based IDS (HIDS) or network-based IDS (NIDS). HIDS is installed at a host to periodically monitor specific system logs for patterns of intrusions. In contrast, an NIDS sniffs the traffic to analyze suspicious behaviors. A *signature-based* NIDS (SNIDS) examines the traffic for patterns of known intrusions. SNIDS can quickly and reliably diagnose the attacking techniques and security holes without generating an overwhelming number of false alarms because SNIDS relies on known signatures. However, *anomaly-based* NIDS (ANIDS) detects unusual behaviors based on statistical methods. ANIDS could detect symptoms of attacks without specific knowledge of details. However, if the training data of the normal traffic are inadequate, ANIDS may generate a large number of false alarms.

This study focuses on the performance of signature-based NIDS because this model is more essential at security gateways [5]. Integrating such NIDS with a firewall is controversial because NIDS will consume too many system resources. However, integrating them together has the potential advantage of enabling an IDS to instantly notify the firewall module to block hostile activities. Several firewall vendors have now incorporated basic anti-DoS functions.

Obviously, the design considerations for an NIDS are:

- Distinguishing intrusions from normal traffic.
- Functioning without consuming too many system resources.
- Performing well even under a heavy traffic load [6].

An example high-performance NIDS [7] proposes a concise and extensible intrusion specification language, and an efficient pattern-matching approach whose matching time is insensitive to the number of patterns.

Network Address Translation

Network address translation (NAT) [8, 9] is a basic requirement of a security gateway. Originally NAT is to save public IP addresses by translating each packet's source private IP address and port number pair into those of the security gateway. In this way, NAT also avoids directly exposing private hosts to the publicly accessible Internet. Thus, NAT can alleviate the danger of being directly attacked. Consequently, many commercial security gateways, including all those to be evaluated later, have built-in NAT functions.

In response to the growth of the above technologies, testing has become essential to verify that they can operate properly and perform well. Additionally, this tutorial seeks to present the experiences of building a product-like integrated security gateway that can support a firewall, VPN, IDS, and NAT from many open-source packages, and determine the performance bottlenecks to suggest future research directions.

The selected open source packages include Linux kernel [10], ipchains [11], Squid [12], FireWall ToolKit (FWTK) [13], FreeS/WAN [14], and Snort [15]. For access control, ipchains, Squid, and FWTK can provide packet filter, URL filter, and content filter capabilities, respectively. For data security, FreeS/WAN provides encryption, authentication, and Internet key exchange services. For intrusion detection, Snort can detect the intrusions in the protected network, and alert system administrators when suspicious activities are detected. Although each package works well individually, they do not cooperate well. Thus, we trace the packet flows in a gateway to find out the problems, and eliminate those problems by modifying the kernel and setting proper rules. We developed the Web management console to manage each module. The highly integrated system is downloadable at [16]. Three special packet flows that require extra integration efforts will be discussed.

[1] *Buffer overflow attack is a common hacking approach that skillfully feeds a long command to overflow the stack of the process. The overflowed segment contains bad actions that are executed when the return address of the stack points to the bad actions.*

Package name	User-space program	Kernel-space program	Package size	Version
ipchains	Command-line management tool	Kernel built-in packet filtering firewall and NAT	63KB	1.3.9
Squid	Daemon (cache server, transparent proxy**, and URL filter)	No	1104KB	2.3
FWTK	Daemon (application proxies for Web content filter)	No	476KB	2.1
FreeS/WAN	Pluto Daemon (Internet key exchange, IKE)	KLIPS kernel patch (encryption and authentication)	1252KB	1.5
Snort	Daemon (intrusion detection)	No	644KB	1.7

* NAT in Linux kernel 2.2 is called IP masquerade (MASQ). For clarity, this article names all NAT services as NAT.
** A transparent proxy automatically redirects Web requests to itself. Users are exempted from explicitly setting their browsers to access the proxy.

Table 1. *Software package information.*

No.	Access types	Demands of protection
1	Untrusted region services (FTP, … except Web)	NAT, and packet filtering
2	Trusted-region communications	VPN, and packet filtering
3	Untrusted region Web services	NAT, packet/URL/content filtering

Table 2. *Three access types.*

After these packages are integrated, a series of external (black-box) and internal (white-box) benchmarks are performed. In external benchmarks, the integrated security gateway is compared with commercial ones to examine its competitiveness, in terms of throughput and latency. In the internal benchmarks, a detailed profiling of the integrated security gateway is performed to examine the CPU/memory/disk consumption and to investigate the scalability of each key module. The questions to be answered include: Who tops the processing time among the kernel-space modules and the user-space processes, respectively? How much disk and memory space does each package consume? How scalable are ipchains, Squid, and FWTK? What are the consequences of increasing the key length in cryptographic algorithms? Can Snort examine each packet for suspicious activities under a heavy traffic load? Where are the bottlenecks of these modules?

The rest of this article is organized as follows. We describe system integration efforts, and then present the external and internal benchmark experiments, respectively. Finally, we give the result and present possible directions for future research.

System Integration

The selected software packages are introduced herein. Some can cooperate well, but some require extra integration efforts. The complete packet flow of our integration is explained in this section.

Selected Packages

Table 1 lists the open-source packages chosen for integration. They are selected because of their functional completeness and excellent reputation. Notably, a Linux system consists of a kernel space and a user space. Kernel space is responsible for abstracting and managing a machine's resources, including process, memory, file system, devices, and networking. User space programs use the kernel-supported system calls. Programs that run permanently as background processes are called daemons.

Extra Integration Works

This section discusses three special access types that require extra integration works. The following section then illustrates the complete packet flows of our integration. Table 2 and Fig. 1a give the three access types and their demands of protection. Connections of type-1 access are established from private hosts to untrusted regions (m1 to F in Fig. 1a), such as public Internet servers, excluding the Web-server accesses that are treated as type-3 accesses. Type-2 connections are *tunneled* between private subnets (m1 to m2 in Fig. 1a) and do not need URL/content filtering since both sides are trusted regions. Type-3 accesses issue HTTP requests from private hosts to public Web servers in the untrusted regions (m1 to W in Fig. 1a) to retrieve Web pages back to the private hosts. These accesses should be protected by various firewall actions shown in Table 2. Figure 1b and c detail the dynamic packet processing flows within VPN1 and VPN2, respectively. Readers can follow the labeled arrow lines to trace the processing flows of the three access types. Figure 1c only illustrates type-2 access because type-1 and type-3 accesses do not enter VPN2. Detailed packet flows of each access type are then described.

Let (X,Y) be the pair of the source IP address X and the destination IP address Y. In Fig. 1b and 1c, the packet flows in VPN1 and VPN2 indicate that the ipchains consists of input, forward, and output chain tables, which are used to check incoming, forwarding, and outgoing packets, respectively. All three chains are checked in sequence when forwarding a packet. In each chain table, the gateway searches for the access rule linearly from top to bottom to match the packet. Each access rule contains the "*condition*" and "*action*" fields to define specific actions taken under specific conditions. Condition defines the packet-matching criteria, such as a certain subnet or application. Action defines the operation imposed on the packets, such as accept, reject, bypass, redirect, or NAT. Any packet matched by a "bypass" rule will be directly skipped to the next chain table, if any, without continuing to search the rest of the table entries in the same chain table. Packets matched by the "redirect" rules are redirected to user-space daemon programs. In this integration ipchains works as a dispatcher that dispatches different packets to their corresponding processing paths. This integration ensures the interoperability among packages so that packets of different

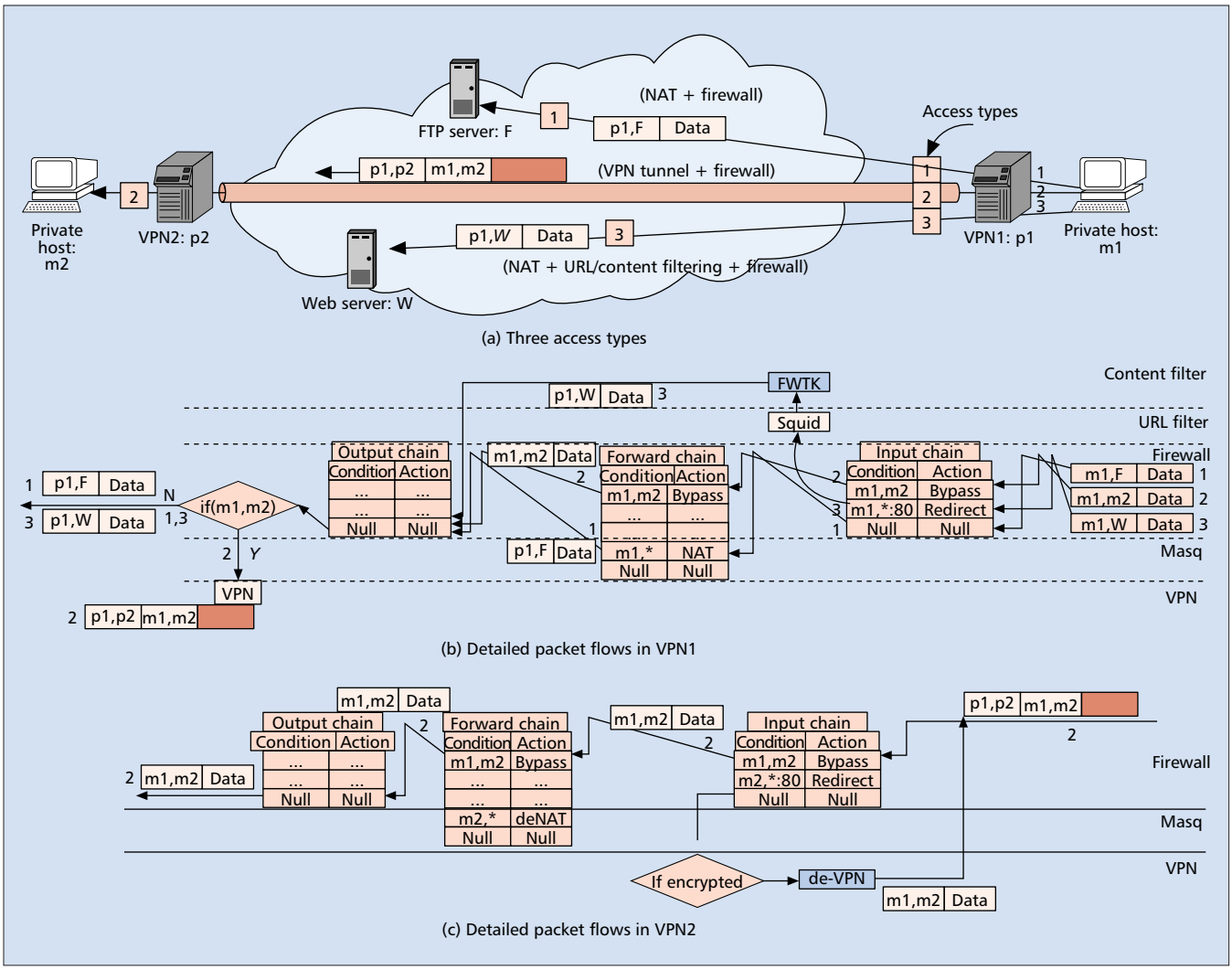

FIGURE 1. *Packet flows of three access types.*

access types can receive appropriate protection. The packet flows of the three access types in Table 2 and Fig. 1 are described below:

Untrusted Region Services (Except Web Service): These services requires NAT and packet filtering protections. The NAT service is a built-in function of ipchains. This integration skillfully avoids directing packets of the IP address pair (m1, F) to the other two access types by sequencing the filters.

Trusted-Region Communications (Brach Offices): Packets of the IP address pair (m1,m2) will first be bypassed in the input chain to avoid being redirected to the path for filtering URL/content. They are bypassed again in the forward chain to avoid being NATed. Finally, they can be encrypted and tunneled. On the other gateway (VPN2 in Fig. 1), any encrypted packets are decrypted, de-tunneled, and looped back to the input chain for packet filtering.

Untrusted Region Web Services: Any HTTP request packet from m1 with IP address pair (m1,W) and destination port 80 will first be redirected to Squid for URL filtering. Subsequently, it is passed to FWTK in Fig. 1b through the inter-daemon socket. FWTK establishes a new connection to W with the IP address pair (p1,W) such that NAT can also be achieved. FWTK then performs content filtering to the retrieved Web pages and forwards them back, if any, to the private host m1 via the original connection.

COMPLETE PACKET FLOWS OF OUR INTEGRATION

Figure 2 illustrates the complete packet processing flows of the integrated security gateway. The flows are traced by kernel debugging tools/techniques: *KProf* [17], *kdb* [18], and *printk*.[2] Figure 2 can be employed at both-side security gateways. In general, the packet flow in a gateway from the private interface to the public interface is as follows:

- Perform checksum calculations and sanity checks of the packet.
- Check the packet through the input chain.
- Route the packet.
- Judge whether the packet is generated by the gateway itself.
- Check the packet through the forward chain.
- NAT the packet, if needed.
- Check the packet through the output chain.
- Perform VPN processing, if needed.

In addition, if the packet's source and destination IP addresses both belong to the gateway, i.e., inter-daemon communication from Squid to FWTK, the gateway passes the packet back to the loopback (lo) interface after checking the output chain.

[2] *Printk is a printf-like C function used in Linux kernel to dump a message to the screen/log.*

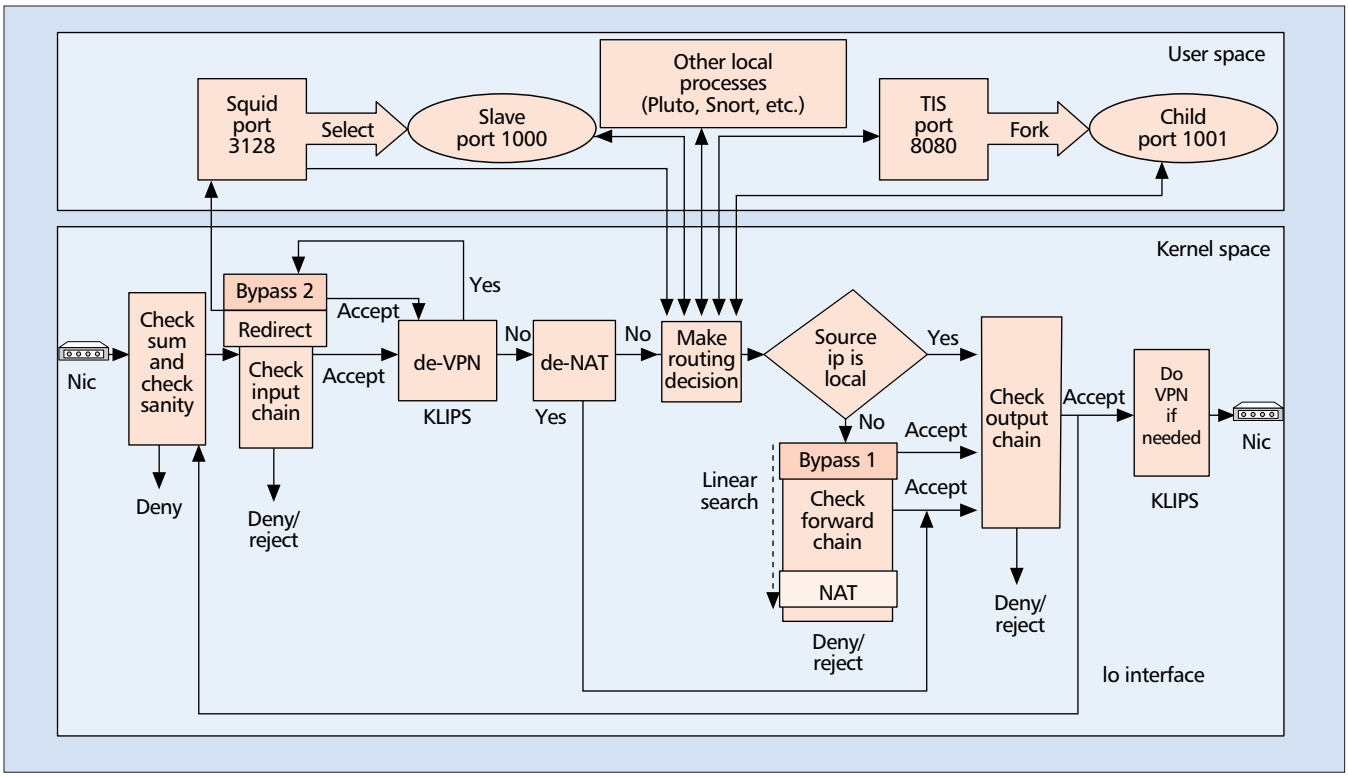

FIGURE 2. *Complete packet processing flows of the integration.*

EXTERNAL BENCHMARK

BENCHMARK METHODOLOGY

This section quantifies the difference in performance between our open-source integration and commercial products. Because vendors tend to give their products a favorable position in the marketplace, some of their announced performance benchmarks confuse users. RFC1242 [19] and RFC2544 [20] then define the benchmarking terminology and methodology, respectively, for network interconnected devices. RFC2647 [21] further defines the benchmarking terminology for firewall performance in response to the strong demand for firewall boxes. Table 3 lists the tested security gateways.

Table 4 describes the benchmark tools, benchmark items, and performance metrics of interest. URL filtering and IDS are not tested because most products do not support them. SmartBits [28] can offer progressively heavier traffic loads until the gateways begin to drop packets. The highest load with zero loss is called the *no loss maximum throughput* (NLMT), and is defined as throughput in RFC2647. WebStone [29] can open as many connections as possible per second to a Web server. Hence it can measure the throughput under the maximum connection rate. When encrypting and tunneling a 1518-byte packet (the maximum Ethernet frame size) from one subnet to the other, the VPN1 gateway requires that the packet be split because encryption and tunneling enlarge the packet size. VPN2 then reassembles it.

In the packet filtering test, the two SmartBits ports emulate two IP subnets. Each port emulates 200 hosts and generates 128-byte and 1518-byte UDP packets from one subnet to the other. The maximum number of packets per second for the two packet sizes with a 100 Mb/s interface reach 97656 and 8234, respectively. Filters are configured not to block

Device under test (DUT)	Solution	OS	CPU	RAM	Flash/HD	Accelerator
Linux 2.2.16(open source)	S/W	Linux	Our PC equipped with P-III 700MHz CPU, 128MB SDRAM, and 15GB hard disk			No
BorderWare [22] Firewall Server v6.1.2	S/W	BSD				No
Check Point [23] VPN/Firewall-1 v4.1 SP 2	S/W	NT				No
Cisco [24] PIX 525R v5.3	H/W	IOS	PIII 600	128 MB	16 MB	No
NetScreen [25] *100 model*	H/W	Proprietary	Galileo 64120 75 MHz	128 MB	128KB	Proprietary ASIC
SonicWALL [26] PRO VX *v5.11*	H/W	Proprietary	Intel StrongARM 233 MHz	16 MB	4 MB	VPN accelerator card
WatchGuard [27] FireBox_Plus	H/W	Linux	K6-III P 366	256 MB	8 MB	No

Table 3. *Devices under test.*

Category	Benchmark tool	Benchmark item	Performance measures
Packet filtering test w/o NAT	SmartBits 2000 governed by SmartFlow 1.2 software	1. Impacts of enabling NAT 2. Impacts of increasing the number of filters	1. NLMT and latency when enabling NAT 2. NLMT and latency when increasing the number of filters
Content filter	WebStone 2.5	1. Impacts of enabling content filter 2. Impacts of increasing HTTP connection rate	Throughput under maximum connection rate
VPN	SmartBits 2000 governed by SmartFlow 1.2 software	1. Impacts of creating a VPN tunnel 2. Impacts of handling packet fragmentation in VPN processing	1. NLMT 2. Latency under VPN tunnel

Table 4. *Categories of external benchmark.*

packets, so SmartBits can accurately trace the NLMT.

In the content filtering test, an Apache 1.3.12 Web server is set up on one side of the gateway, and WebStone acts as a Web client on the other side. The Web pages obtained from the WebStone package do not contain any ActiveX/Java/JavaScript objects. Therefore, all devices under tests (DUTs) are only to check whether any ActiveX/Java/JavaScript objects are present in the retrieved Web pages.

Figure 3 presents the benchmark environment of the subnet-to-subnet VPN in the IPsec VPN test. Both-side DUTs are connected to each other via a wire-speed layer-3 switch. The configurations of SmartBits are the same as those for packet filtering. Both gateways are set to operate in IPsec's ESP (Encapsulated Security Payload) mode [30] using 3DES (Triple Data Encryption Standard) encryption [31] and MD5 (Message Digest Algorithm) authentication algorithms [32].

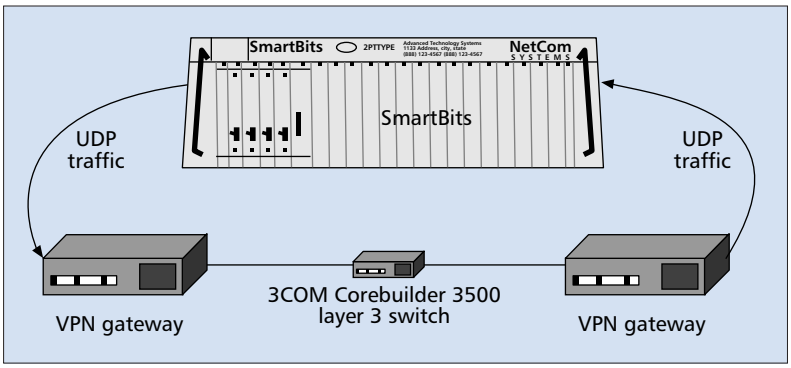

FIGURE 3. *Environment for Evaluating VPN throughput.*

PERFORMANCE RESULTS

Packet Filter and NAT — Packet filtering only examines packet headers and hence is sensitive to the *packet count* per second, which is determined by the *offered load* and the *packet size*. The implementation mechanisms and hardware specifications affect NAT performance. Fig. 4 compares the NLMT, with or without NAT, among the 7 DUTs. BorderWare cannot disable NAT, and SonicWall was returned before the results without NAT were collected. Although the hardware platforms of the three software-based solutions are the same, their performance differs considerably. BorderWare is an application-proxy firewall and therefore has the worst performance. Check Point's solution may be bottlenecked by the Windows NT operating system. Open-source solution performs well (86.6 Mb/s) for 128-byte packets but is significantly degraded (57.8 Mb/s) when NAT is enabled. In contrast, NetScreen's ASIC exhibits wire-speed throughput.

The NLMT of CheckPoint, SonicWall, and WatchGuard are similar (Fig. 4), but the latency of SonicWall is low, as depicted in Fig. 5. CheckPoint and WatchGuard are inferred to use a much larger buffer to queue packets than is used by Son-

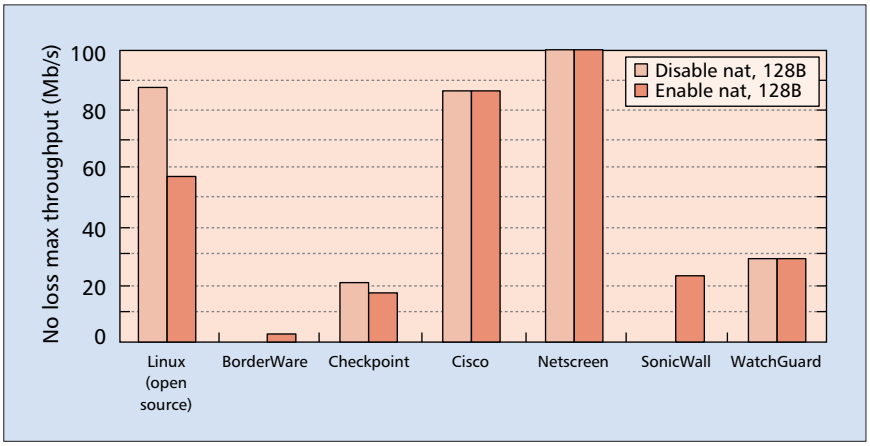

FIGURE 4. *No loss maximum throughput with/without NAT enabled.*

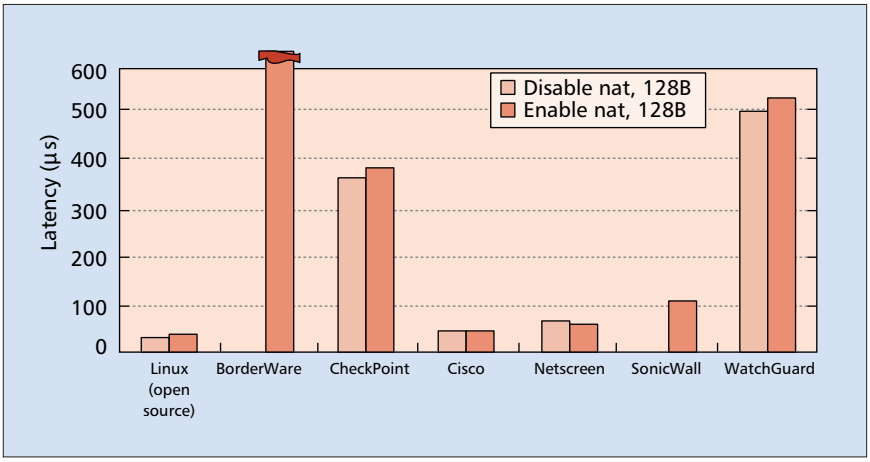

FIGURE 5. *Latency with/without NAT enabled (traffic load 10 Mb/s).*

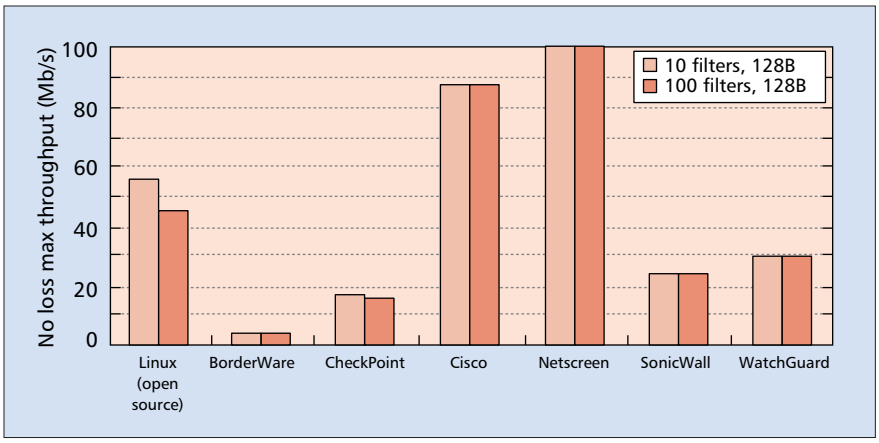

FIGURE 6. *No loss maximum throughput under 10/100 filters with NAT.*

icWall. In contrast, Fig. 6 reveals that, for 128-byte packets, the open-source solution can reach 55.7 Mb/s and 45.1 Mb/s with 10 filters and 100 filters, respectively. The differences between the results with 10 filters and those with 100 filters lie in the linear search against the filter database. The previously obtained experimental results[3] showed that, for 1518-byte packets (packet count reduced to 8.4 percent of that for 128-byte packets) the benchmark results are much better than those for 128-byte packets. For 1518-byte packets, all DUTs except BorderWare can achieve 90 Mb/s.

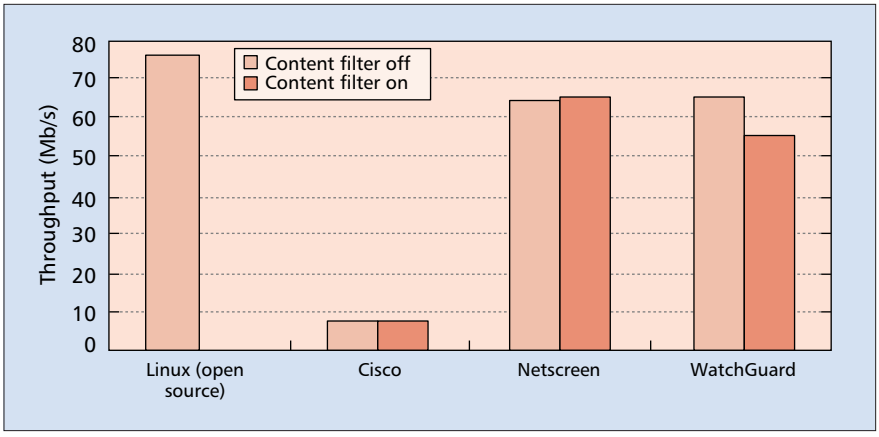

FIGURE 7. *Throughput at maximum connection rate.*

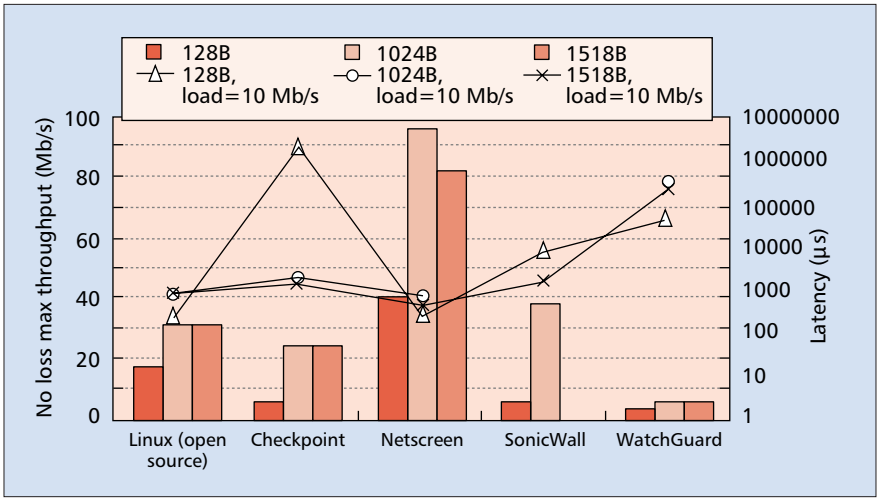

FIGURE 8. *No loss maximum throughput of subnet-to-subnet VPN tunnel.*

Content Filter — Content filtering requires the packet to be scanned, and can thus be sensitive to both packet count and packet size. Figure 7 compares the throughput of content filtering under many HTTP connections. Notably, not all DUTs support complete content filtering. NetScreen and WatchGuard are only aware of the "content-type" field in the HTTP header. Thus they can only filter out ActiveX and Java objects. In contrast, the open-source FWTK and Cisco solutions can scan the entire retrieved Web page and filter out unwanted HTML tags, such as embedded JavaScripts. However, open-source FWTK has a very low throughput. The later section on internal benchmarking further investigates the cause.

VPN — Packet size matters for VPN because the packet payload must be encrypted/decrypted and authenticated. As Fig. 8 shows, only five of them support the 3DES encryption and MD5 authentication algorithms. CheckPoint delivers less than the open-source FreeS/WAN. NetScreen and SonicWall offload the CPU-intensive processing to their ASIC and accelerator card, respectively. The latencies of FreeS/WAN and NetScreen are similar, but the NLMT of NetScreen is much greater. However, for 1518-byte packets, the NLMT of NetScreen falls because of the required handling of packet fragmentation, as described in an earlier section.

In general, the open-source solution gives competitive performance. However, the content filter, FWTK, suffers from serious degradation. The bottlenecks of each module are identified below using white-box internal benchmarking and code tracing.

INTERNAL BENCHMARK

BENCHMARK TOOLS AND METHODOLOGY

To further identify the bottlenecks of the open-source solutions, we conduct a series of internal benchmark experiments, as depicted in Table 5. SmartBits can generate UDP packets with sizes ranging from 64 bytes to 1518 bytes. A self-developed HTTP traffic generator instead of WebStone is used to generate HTTP requests to retrieve specific Web pages of different sizes. The CPU/memory/disk consumptions and scalability of each key module is investigated. Time-stamps are inserted before and after each module. The time-stamps are taken by the x86 RDTSC instruction to read

[3] *1518-byte test results among DUTs are similar (all above 90 Mb/s except BorderWare) so we omit it.*

Category	Benchmark tools	Settings	Benchmark items
CPU cost	SmartBits 2000 with SmartFlow 1.2, self-written HTTP traffic generator	1. Enable all security functions 2. No other filters in ipchains 3. Using 3DES/MD5 4. 10 URL entries are configured	Who tops the processing time among all the functions
Memory and disk cost	SmartBits 2000 with SmartFlow 1.2, self-written HTTP traffic generator	5. A single HTTP connection repeatedly retrieves a 40KB Web page for 10 seconds	The disk/memory consumptions
Packet filter	SmartBits 2000 with SmartFlow 1.2	1. Only enable packet filter 2. Various numbers of filters 3. Various packet sizes	Scalability
URL filter	Self-written HTTP traffic generator	1. Various Web page sizes 2. Various URL lengths in HTTP request 3. A single HTTP connection repeatedly retrieves a 64KB Web page for 10 seconds	Scalability
Content filter	HTTP traffic generator	1. Various Web page sizes 2. Various numbers of concurrent connections 3. A single HTTP connection repeatedly retrieves a 64KB Web page for 10 seconds	Scalability
IP masquerade	SmartBits 2000 with SmartFlow 1.2	1. Security gateway equipped with 4 NICs, one for public interface and three for private interfaces 2. Various numbers of private hosts 3. Various packet sizes	Scalability
Authentication algorithms	SmartBits 2000 with SmartFlow 1.2	Various packet sizes	Cost of MD5 and SHA1
IDS	SmartBits 2000 with SmartFlow 1.2	Various packet sizes	1. Packet loss rate 2. Pattern-matching time

Table 5. *Benchmark methodology.*

the 64-bit register that increments at every clock cycle. The accuracy achieves 1000/700 ns when using the P-III 700MHz CPU. The test results are saved in variables, and read off-line after each experiment. Thus the overheads are very low. In resource consumption experiments, all security functions are enabled but only the function that is being investigated is time-stamped before and after the module to avoid overhead. Unix tools, top and ps, are used to examine the memory and disk consumptions.

Consumption of Resources

CPU Time Consumption — Figure 9 quantifies the CPU cost of each kernel module using 64-byte packets. The 0 Mb/s traffic load refers to the scenario without background traffic, and 13 Mb/s is the NLMT of the gateway when all the functions are enabled. For a 64-byte packet, the 3DES encryption takes 24.242 s, which is four times that of the MD5 authentication, and 12 times that of NAT. From other experimental results, the processing time for encryption, authentication, and NAT depends on the packet size because these modules process each entire packet. Notably, the NAT process recalculates the transport layer checksum. For a 1518-byte packet, the 3DES encryption takes 287.983 s, which is nine times the time required by the MD5 authentication, and 31 times that required by NAT. Encrypting 24, or approxi-

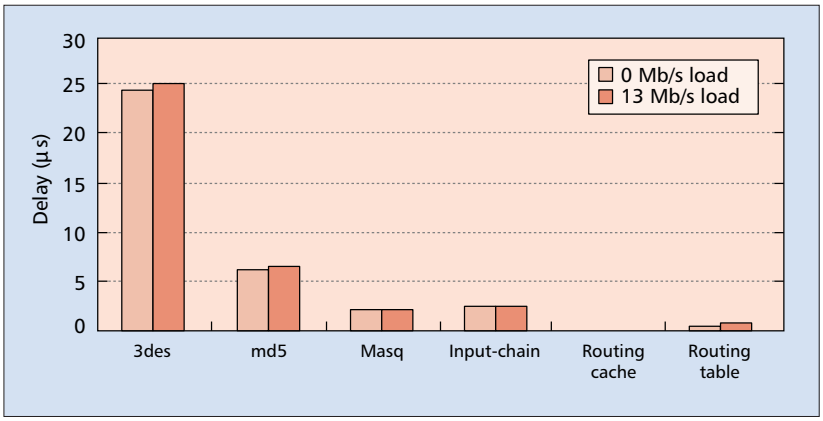

Figure 9. *CPU cost of kernel modules.*

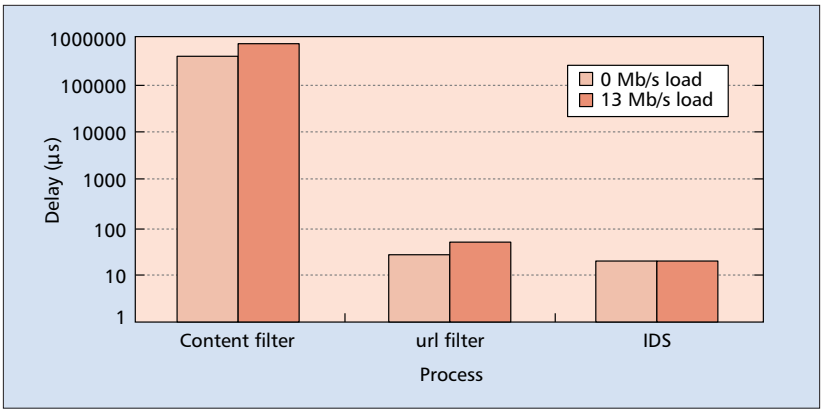

Figure 10. *CPU cost of daemon processes.*

Module	Program size	Swap memory	Resident memory
Kernel	640 KB		2056 KB
Squid Parent	468 KB	3348 KB	880 KB
Squid Child		13544 KB	12092 KB
FWTK http-gw parent	1788 KB	576 KB	200 KB
FWTK http-gw child		1708 KB	668 KB
Pluto Daemon	646 KB	1516 KB	716 KB
Snortd	444 KB	3236 KB	2268 KB

Table 6. *Memory and disk consumption.*

mately 1518/64, 64-byte packets requires 581.808 s, which is twice as much as the time to encrypt a 1518-byte packet. This is because a 1518-byte packet requires only one encryption operation, while 24 64-byte packets require 24 encryption operations.

Figure 10 shows the CPU cost of each daemon process. Again, the content filter FWTK has some design problems that will be identified later.

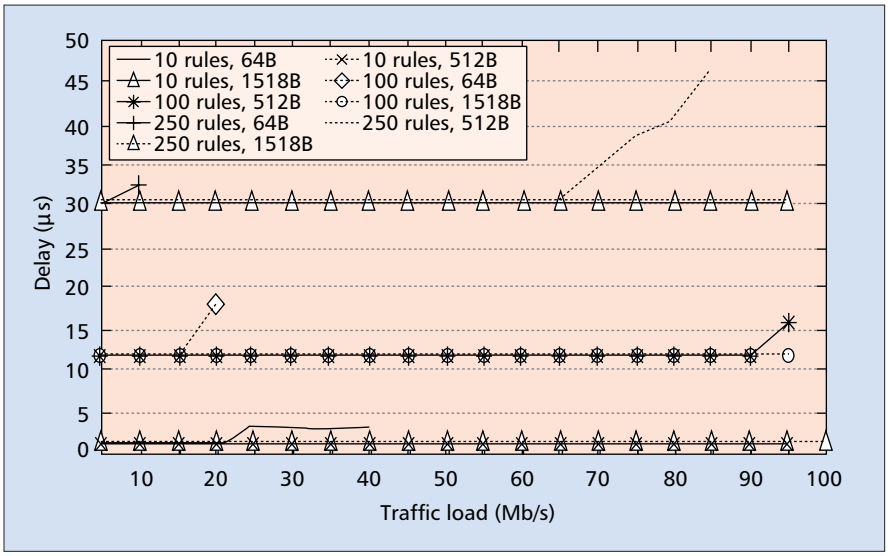

FIGURE 11. *. Scalability of packet filter in ipchains: each test case is performed until the gateway begins to drop packets.*

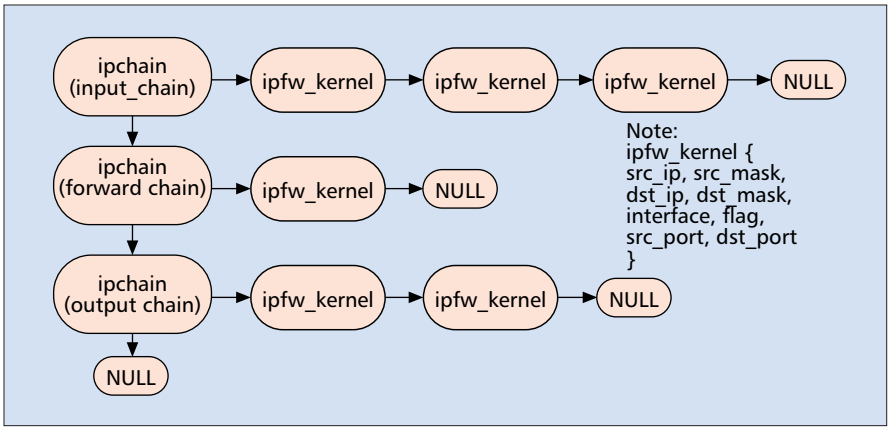

FIGURE 12. *Data structure of packet filter in ipchains.*

Memory and Disk Consumption — Table 6 summarizes the memory and disk consumption of each module. The swap and resident memory shows the run-time requirements of disk space and physical memory, respectively. Squid consumes a total of 17.3MB of disk space and 12.9MB of memory, mainly because of Web caching.

Scalability Issues

Packet Filtering — Figure 11 plots the latency of the input chain module under various numbers of rules and packet sizes. The processing times required to check the input chain under light load are 1.6 s for 10 filters, 11.9 s for 100 filters, and 30.3 s for 250 filters. As the load increases, the delay rises considerably mainly due to interrupts generated by incoming packets. Figure 12 illustrates the data structure of ipchains. When a packet enters the input chain, the gateway *linearly* checks the table entries (organized as linked lists) of the input chain until one entry is matched or the end of the list is traversed. Consequently, the delay is highly dependent on the number of filters. The worst-case time complexity of ipchains is $O(l + m + n)$, where l, m, and n stand for the number of filters in the input, forward, and output chains, respectively.

URL Filter — According to Fig. 13, the number of regular expressions of URL and the URL's length in the HTTP requests are the dominant factors. Squid maintains the regular expressions of URL using a linked list for the *linear* matching process. Each character of the configured URL and the request URL is scanned only once. Thus, the worst-case time complexity of pattern matching is $O(l + m)$, where l and m stand for the URL's length in the HTTP requests, and the average length of regular expressions, respectively. Finally, the worst case time complexity of URL filtering in Squid is $O(n(l + m))$, where n represents the number of URL regular expressions.

Content Filter — According to Fig. 14, the mean filtering time for 500K-byte Web pages is 68.235ms under 15 concurrent connections, which is not scalable. Further source-code tracing of FWTK (Fig. 15) reveals two implementation problems. First, FWTK is found to *fork* a child process to tackle every incoming HTTP request. Moreover, each child process re-reads the configuration file, which involves slow disk access. Second, FWTK performs the filtering service using a Finite State Machine (FSM), and FWTK reads just one byte of the Web page at a time from the socket interface to drive its FSM. This implementation is inefficient. The worst case time complexity of FWTK is $O(n)$, where n represents the size of the retrieved Web page.

Instead of reading one byte of the Web page at a time, reading multiple bytes at once can reduce n.

NAT — In Fig. 16, the NAT processing time under 2997 clients is 106.81 s for 64-byte packets, and 118.248 s for 512-byte packets. Three major tasks of NAT processing are checking out the NAT table, rewriting the packet's header, and recalculating the packet's checksum. However, the latter two tasks are not time-consuming. The three tasks consume a total of 1.75 s for 64-byte packets and 3.17 s for 512-byte packets.

As illustrated in Fig. 17, NAT processing uses two hash tables to maintain the mapping between private and public hosts. Each table consists of 256 hash buckets and many doubly linked lists. When one new connection passes through the gateway, the gateway creates a new NAT entry into each table. The gateway checks out the ip_masq_s_table and the ip_masq_m_table for outgoing and incoming packets, respectively. The worst case time complexity of NAT is $O(n)$, where n represents the number of the private-to-public connections. This NAT scales well.

MD5 and SHA1 Authentication Algorithms — This internal benchmark includes neither DES nor 3DES tests because FreeS/WAN does not support DES algorithms. FreeS/WAN implements HMAC-MD5 and HMAC-SHA1 to authenticate and determine the integrity of the data. The HMAC structure in Fig. 19a can enhance the cryptographic strength of its embedded hash algorithm, such as MD5 and SHA1. The MD5 and SHA1 algorithms are quite similar since they are both derived from the MD4 algorithm. Their main difference is that the SHA1 digest is 32 bits longer than the MD5 digest. Accordingly, HMAC-SHA1 executes more slowly than HMAC-MD5 on the same hardware, as shown in Fig. 18. The time for processing 1518-byte packets is 31.89 s and 79.84 s, for MD5 and SHA1, respectively. Figure 19b illustrates the digest generation process:

- Append padding bits to the original message.
- Append the length of the original message.
- Initialize input key.
- Process the message in a sequence of 512-bit blocks.
- Generate the output digest.

Therefore, as the packet size increases, the time for digest generation is extended. The worst case time complexity of these two authentication algorithms is $O(n^*m)$, where m and n represent the key length and the packet size, respectively.

Intrusion Detection System — Figure 20 displays the positions of the Snort and related kernel modules. The sk_buff structure encapsulates each packet in the kernel. The Linux socket filter is used to collect *copies* of packets of interest from the packet buffer to the sock buffer. Thus, packets fed to Snort follow the sniffed packet flow in Fig. 20. The ordinary packets pass through the TCP/IP stack (according to the normal packet flow in Fig. 20) as usual. However, Snort copies one packet at a time from the kernel space to the user space and reassembles them to check for suspicious activity. The checking process as described above is complex and time consuming. Thus, the sock buffer tends to overflow when the traf-

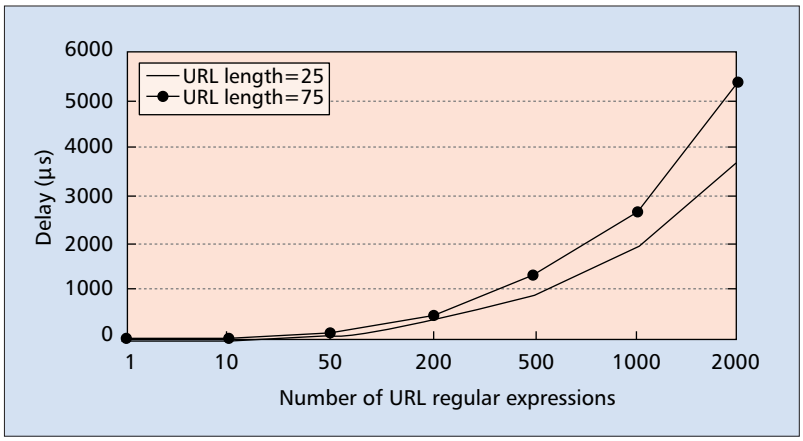

FIGURE 13. *Scalability of URL filter in squid.*

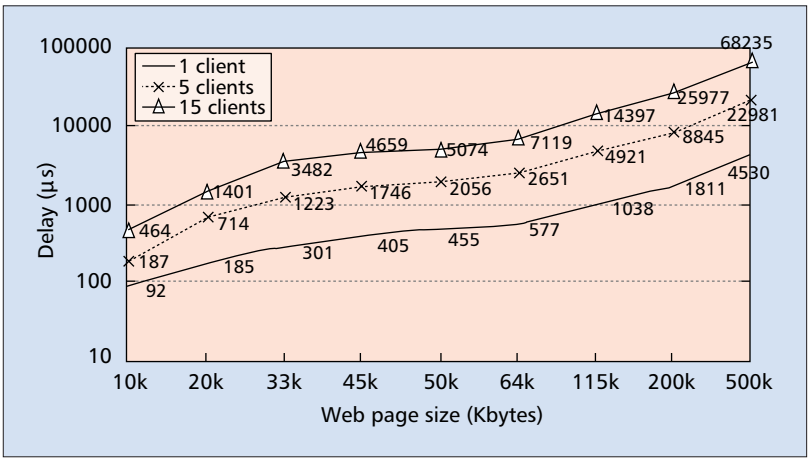

FIGURE 14. *Scalability of content filter in FWTK.*

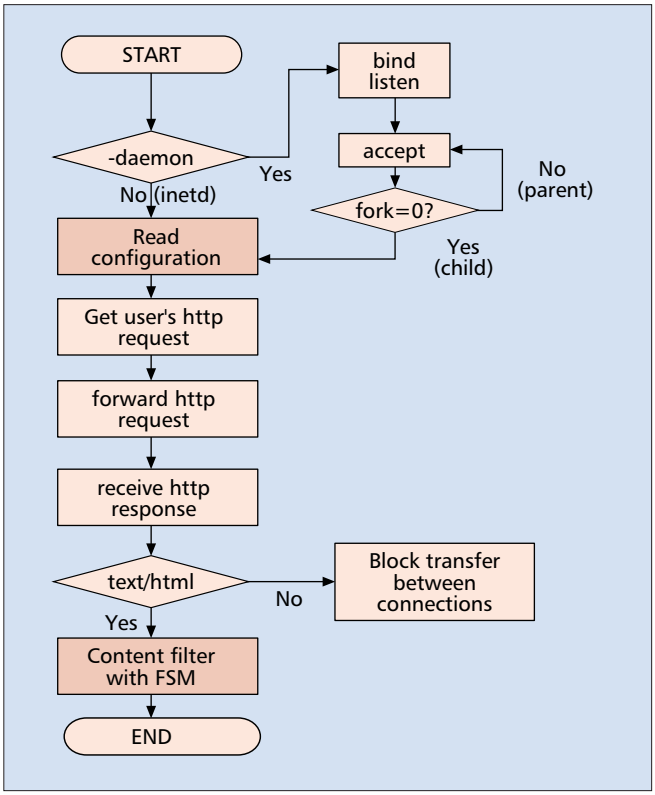

FIGURE 15. *Program flow of content filter in FWTK.*

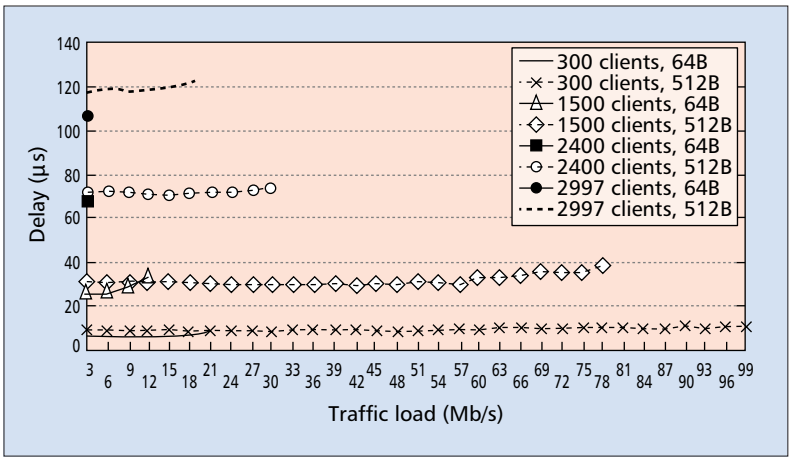

FIGURE 16. *Scalability of NAT: (1) Each port of SmartBits can emulate 999 hosts. Thus, the testbed (three ports for pumping traffic and one port for receiving packets) can emulate 2997 private hosts simultaneously sending packets to 999 public hosts through the gateway. (2) Each test is performed until the gateway begins to drop packets.*

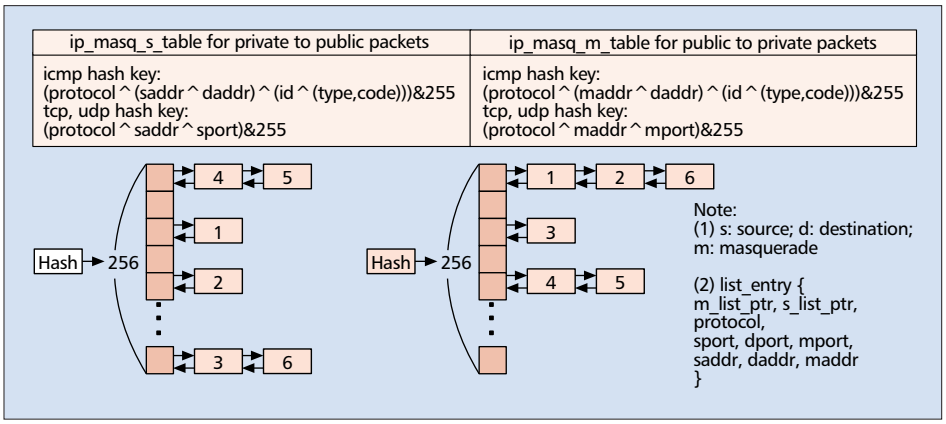

FIGURE 17. *Data structure of NAT table.*

fic load is high. Thus, Snort would miss checking those lost packets. Note that while the loss rate of the *copied* packets for Snort is so high, the kernel still forwards *original* packets without loss.

Figure 21 plots the percentage of packets that Snort cannot check, i.e., those that are lost, when the NLMT is increased. The 64-byte packet loss rate is 85.27 percent under the 30 Mb/s trafic load and 99.86 percent under the 40 Mb/s traffic load. Snort appears to only sample a few packets for checking, allowing intruders to insert intruding packets between normal packets. Fig. 22 details the intrusion pattern-matching time for Snort to check a packet. Under a light load, packet size dominates the processing time, while under a high load packet count per second dominates the processing time.

Figure 23 depicts the data structure of the pattern-matching in Snort. Five separate chains of rules — active, dynamic, alert, log, and pass — are in Snort. Each chain contains three separate linked lists that correspond to TCP, UDP, and ICMP rule-sets. By default, Snort only uses the alert rule chain. It generates an alert using the selected alert method and logs the packet when a suspicious activity is detected. When receiving an UDP packet, Snort follows the links to the alert rule chain, and then checks the packet against its UDP rule-set. Note that in the IDS tests, the rule-set is composed of 49 ICMP rules, 315 TCP rules, and 69 UDP rules. On the average, UDP packets are checked through 23 UDP rule tree nodes and 20 UDP rule options. The worst case time complexity of Snort is $O(l + m^*n)$, where l, m, and n stand for the number of TCP/UDP/ICMP rule tree nodes, the number of TCP/UDP/ICMP rule options, and the packet size, respectively.

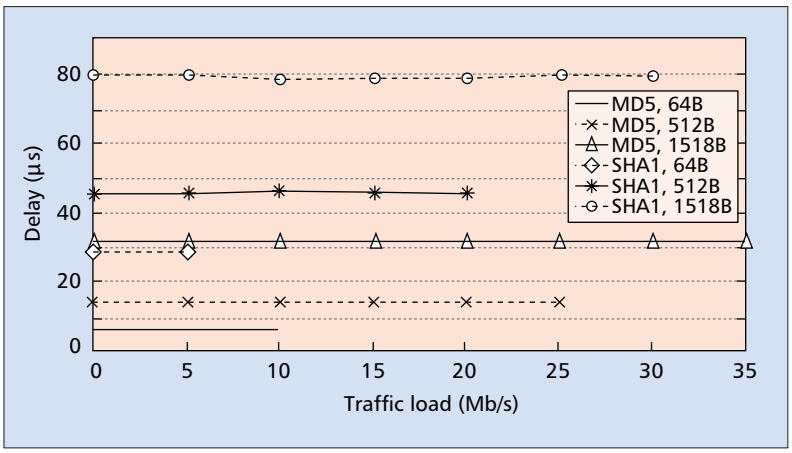

FIGURE 18. *Cost of HMAC-MD5 and HMAC-SHA1 Authentication Algorithms: Each test is performed until the gateway begins to drop packets: a) The HMAC structure; b) MD5/SHA-1 algorithm.*

CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

This work presents the experiences of integrating many open-source packages into a security gateway. The conflicts among the software components are resolved. The external benchmark compares this open-source integrated system with commercial security gateways. The internal benchmark examines the CPU/memory/disk consumption of our integration and investigates the scalability of each key module. Finally, observations concerning the benchmarking and suggestions for improving performance are presented.

Table 7 summarizes the observations of the benchmark results. It indicates that ipchains and FreeS/WAN are more viable than commercial products, but FWTK and Snort have performance problems.

RESEARCH ISSUES FOR PERFORMANCE ENHANCEMENT

The following improvements are suggested to scale up these packages:

Improving the Linear-Matching Algorithms:

For ipchains, Squid, and Snort, their linear-matching algorithms can be accompanied by a flow cache so that active flows can follow a fast path. Typically, the first packet of a flow will traverse through the normal linear-matching phase and establish a flow state, but its following packets can match the flow state by hashing plus linked list (such as that in the NAT module). For stateful inspection firewalls, of course it has incorporated such mechanisms. For signature-based IDS such as Snort, most of the signatures (545 of 763 signatures) attack Web servers. By carefully caching the valid URLs, normal URL accesses can bypass the long linear-matching phase of URL-related signatures.

Proper Implementation Tricks: For FWTK, the configuration file should be scanned only once, and the retrieved Web page can be read multiple bytes at once from the kernel space to the user space. For the NAT module, a suitable bucket size for large enterprises can be defined to avoid hash collisions.

Function Relocation from Daemon to Kernel: For advanced access control policies used in the application proxy, such as FTP, SMTP proxy in FWTK, only the control-plane parts are required to be directed to the user-space daemon process for checking. Other data-plane objects should pass directly through the kernel or be blocked, according to the access-control policy. For example, to prevent employees from downloading MP3 files using FTP, the FTP application proxy needs only to check the FTP-cmd channel for MP3 file extensions and leave the FTP-data channel unchecked. Otherwise, even when nobody downloads the MP3 files, the application proxy still has to relay all FTP sessions, from the kernel to the user space and back to the kernel, to enforce the policy. As another example, some security gateways can be set to append an extra filename extension to some types of email attachments transmitted using SMTP/POP3/IMAP protocols, such that users are alerted to open the file with care. Obviously only the packets that constitute the filename should be directed to the daemon process for matching and modifying. The following packets, which constitute the attached objects, can pass through the gateway if permitted without any processing. Several works have focused on changing the slow kernel-daemon-kernel data path into a fast-kernel data path [33–35]. The efforts differ primarily in the flexibility to switch between slow and fast data paths. Numerical results indicate that this pure software-based acceleration of application proxies can improve the performance by a factor of two to four.

For a signature-based IDS such as Snort, some crucial patterns of intrusions can be moved to the kernel. For example, the most common packet-level intrusions, such as denial-of-service (DoS) attacks, can be in-kernel detected/blocked.

Hardware Accelerators: For encryption/decryption operations in VPN processings such as that in FreeS/WAN, the 3-DES operations can be offloaded to

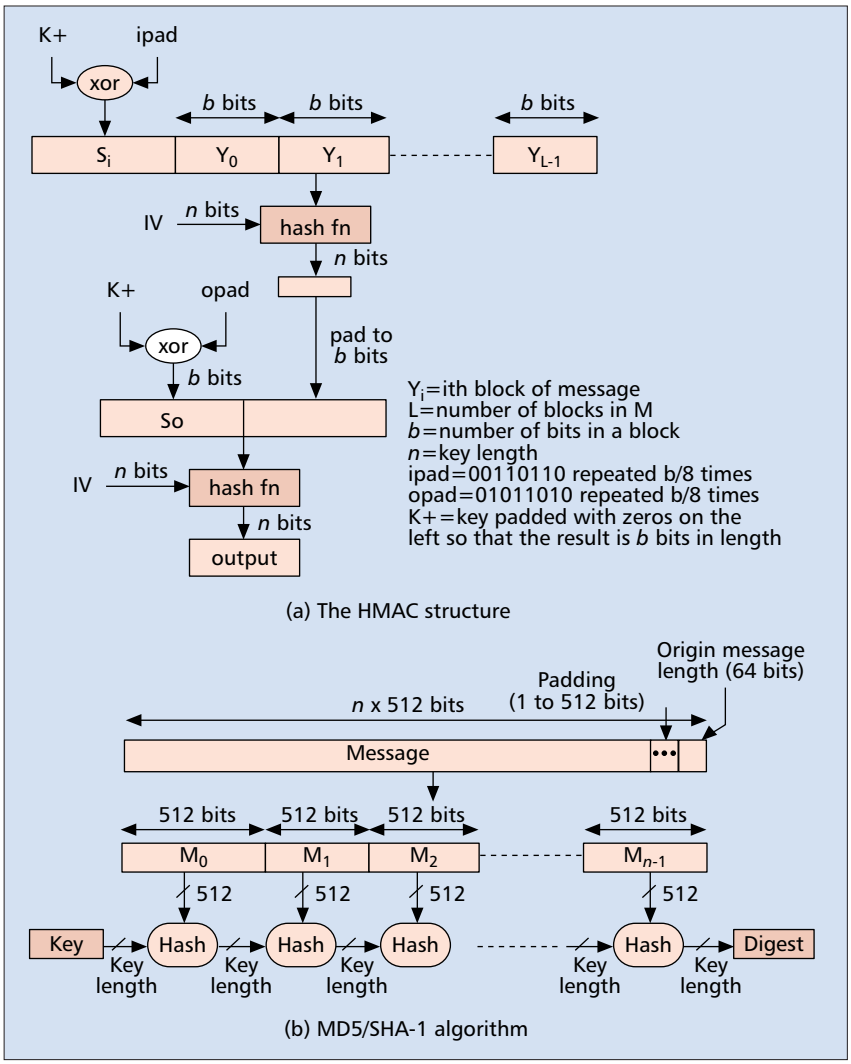

FIGURE 19. *HMAC-MD5 / HMAC-SHA1 digest generation*

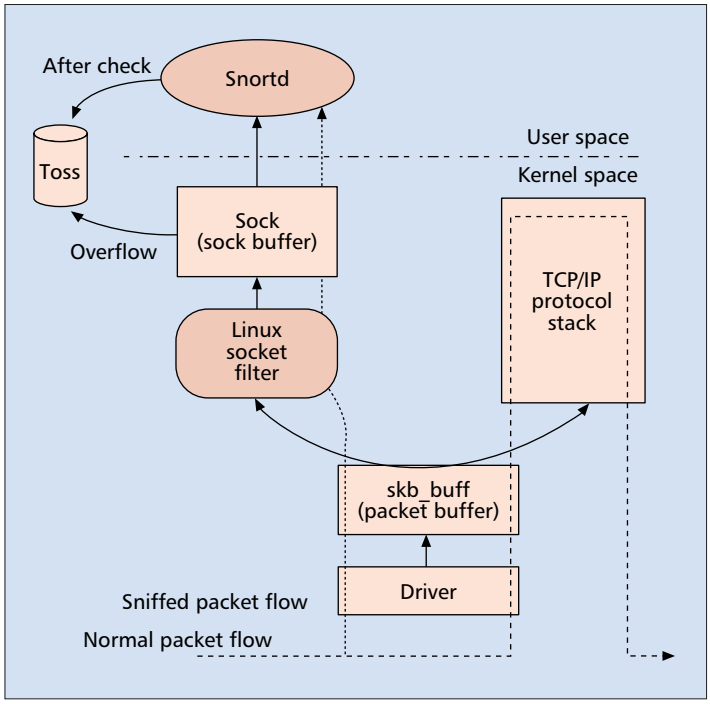

FIGURE 20. *Position of snort.*

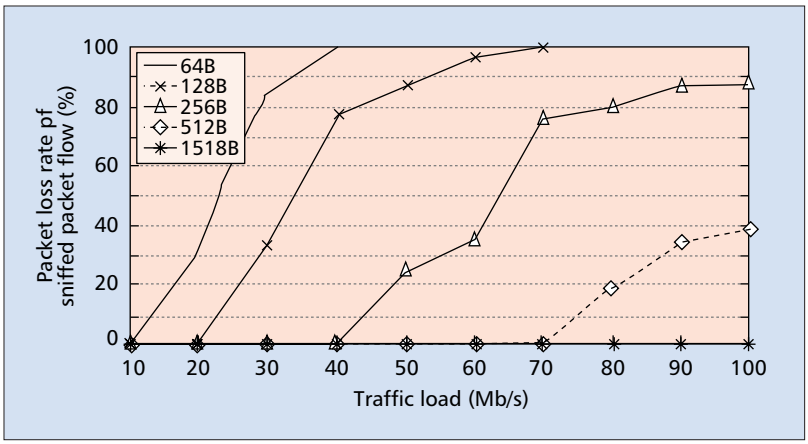

FIGURE 21. *Percentage of missed packets.*

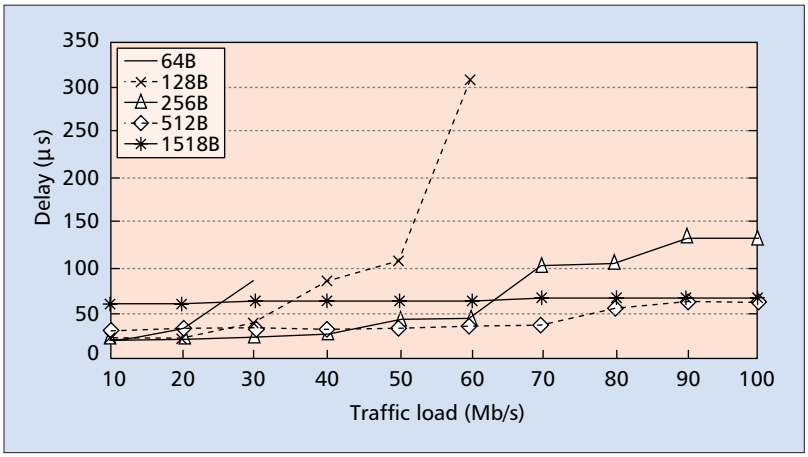

FIGURE 22. *Time for pattern-matching in snort.*

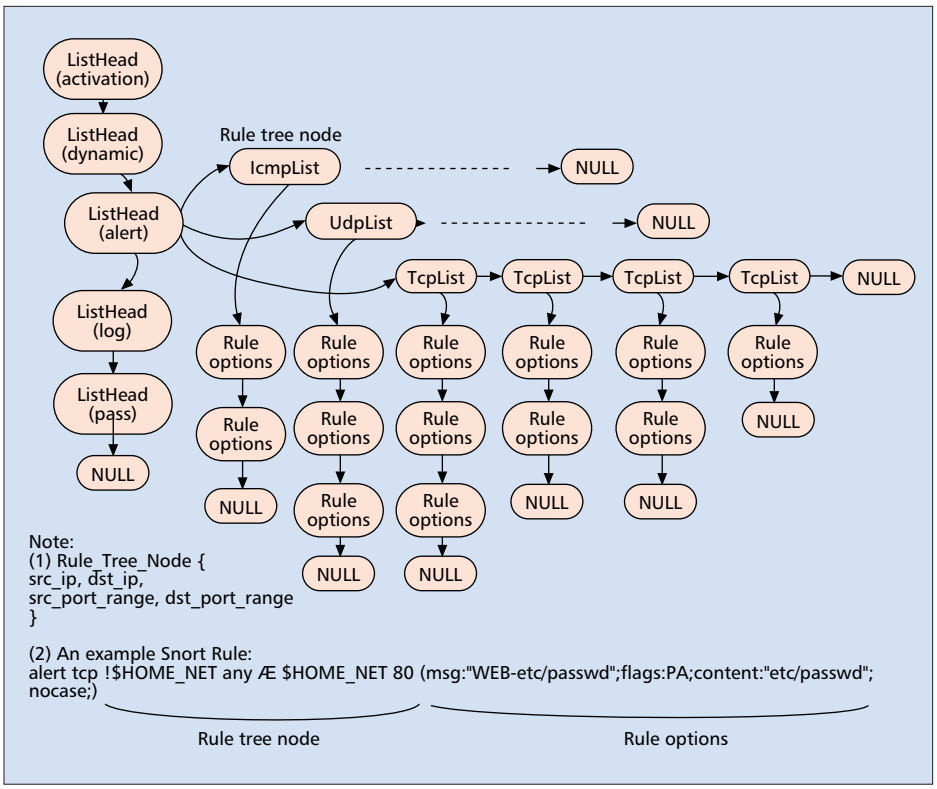

FIGURE 23. *Data structure of pattern-matching in snort.*

an accelerator card or ASIC. Typical operations are:

- Formatting the data to be encrypted/decrypted.
- Feeding data to the hardware through programming I/O or DMA channels.
- Waiting for a hardware interrupt to trigger the Interrupt Service Routine (ISR) to check what is happening.
- Finding that the hardware has successfully encrypted/decrypted.
- Continuing to process the subsequent operations.

Integrating many functions into a single all-in-one system or separating them as standalone devices involves security and performance issues. Many commercial security gateways choose to be an all-in-one solution. Accordingly, this study focuses only on building a product-like, all-in-one system from numerous open-source packages and on externally and internally evaluating the performance of such a system. However, installing such a device does not mean secured. Other issues, such as correctly setting the administrative policy rules, increasing the security of the network architecture, and increasing the security of the encryption algorithms, are beyond the scope of this study and deserve further attention. The highly integrated system presented here, together with the self-developed Web management console, is downloadable at [16] for hands-on practice.

REFERENCES

[1] W. Simpson, "IP in IP Tunneling," RFC 1853, Oct. 1995.

[2] K. Hamzeh *et al.*, "Point-to-Point Tunneling Protocol (PPTP)," RFC 2637, July 1999.

[3] W. Townsley *et al.*, "Layer Two Tunneling Protocol (L2TP) ," RFC 2661, Aug. 1999.

[4] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, Nov. 1998.

[5] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network Intrusion Detection," *IEEE Network*, vol. 8, no. 3, May-June 1994, pp. 26–41.

[6] N. J. Puketza, K. Zhang, and M. Chung, "A Methodology for Testing Intrusion Detection Systems," *IEEE Trans. Software Eng.*, vol. 22, no. 10, Oct. 1996, pp. 719–29.

[7] R. Sekar *et al.*, "A High-Performance Network Intrusion Detection System," *Proc. 6th ACM Conf. Comp. and Commun. Security*, no. 1-4, Nov. 1999, pp. 8–17.

[8] K. Egevang and P. Francis, "The IP Network Address Translator (NAT)," RFC 1631, May 1994.

[9] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Jan. 2001.

[10] Linux kernel, http://www.kernel. org

[11] ipchains, http://netfilter.filewatcher.org/ipchains/

[12] Squid, http://www.squid-cache.org

Module	Characteristics	Bottleneck	Reason	Worst-case time complexity
ipchains	CPU-intensive	Increasing the number of filters	Linear matching algorithm	$O(l + m + n)$; l, m, n: number of filters in input, forward, and output chains, respectively
Squid	Memory- and CPU-intensive	Increasing the number of URL regular expressions	Linear matching algorithm	$O(n(l + m))$; l: URL length in HTTP requests; m: average regular expression length; n: number of URL regular expressions
FWTK	CPU-intensive	Increasing the number of HTTP connections and the size of the retrieved Web page	1. Parse config file for each request 2. Read only one byte of the Web page from the socket interface at a time	$O(n)$; n: size of the retrieved Web page
NAT	CPU-intensive	Increasing the number of private-to-public connections	Data structure of NAT table	$O(n)$; n: number of private-to-public connections
FreeS/WAN	CPU-intensive	Using the stronger algorithms	Too many computations for encryption and authentication	$O(n*m)$; m: key length; n: packet size
Snort	CPU-intensive	Packet loss frequently	1. Copy each packet from kernel space to user space 2. Linear matching algorithm	$O(l + m*n)$; l: number of TCP/UDP/ICMP rule tree nodes; m: number of TCP/UDP/ICMP rule options; n: packet size

Table 7. *Summary of comparison.*

[13] FWTK, http://www.fwtk.org
[14] FreeS/WAN, http://www.freeswan.org
[15] Snort, http://www.snort.org
[16] "Integrated security gateway," http://speed.cis.nctu.edu.tw/SG.html
[17] KProf, http://kprof.sourceforge.net
[18] Using the KDB kernel debug program, http://www.rz.uni-hohenheim.de/betriebssysteme/unix/aix/aix_4.3.3_doc/ext_doc/usr/share/man/info/en_US/a_doc_lib/aixprggd/kernextc/kdb.htm
[19] S. Bradner, "Benchmarking Terminology for Network Interconnection Devices," RFC 1242, July 1991.
[20] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," RFC 2544, Mar. 1999.
[21] D. Newman, "Benchmarking Terminology for Firewall Performance," RFC 2647, Aug. 1999.
[22] BorderWare, http://www.borderware.com
[23] CheckPoint, http://www.checkpoint.com
[24] Cisco, http://www.cisco.com
[25] NetScreen, http://www.netscreen.com
[26] SonicWALL, http://www.sonicwall.com
[27] WatchGuard, http://www.watchguard.com
[28] SmartBits, http://www.netcomsystems.com
[29] WebStone, http://www.mindcraft.com
[30] R. Atkinson, "IP Encapsulating Security Payload (ESP)," RFC 1827 Aug. 1995.
[31] P. Karn, P. Metzger, and W. Simpson, "The ESP Triple DES Transform," RFC 1851, Sept. 1995.
[32] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, Apr. 1992.
[33] D. Maltz, "TCP Splicing for Application Layer Proxy Performance," IBM Research Report, Mar. 1998.
[34] O. Spatscheck *et al.*, "Optimizing TCP Forwarder Performance," *IEEE/ACM Trans. Net.*, vol. 8, no. 2, Apr. 2000.
[35] S. K. Adhya, "Asymmetric TCP Splice: A Kernel Mechanism to Increase the Flexibility of TCP Splice," master thesis at Dept. of C.S., Indian Institute of Technology, Apr. 2001.

Additional Reading

[1] M. Wu and Y. Lin, "Open-Source Software Development: An Overview," *IEEE Computer*, June 2001.

Biographies

Ying-Dar Lin (ydlin@cis.nctu.edu.tw) received his M.S. and Ph.D. degrees in computer science from UCLA in 1990 and 1993, respectively. He was a technical staff member at IBM Taiwan and Bell Communications Research. Since 1999 he has been a professor at National Chiao Tung University in Taiwan. His research interests include the design, analysis, and implementation of network protocols and algorithms, quality of services, network security, and content networking. He is a member of ACM and IEEE. He is the founder and head of Network Benchmarking Lab (NBL). He can be reached at ydlin@cis.nctu.edu.tw and http://www.cis.nctu.edu.tw/~ydlin.

Huan-Yun Wei (hywei@cis.nctu.edu.tw) is a Ph.D. candidate in computer and information science at National Chiao Tung University. His interests include TCP rate shaping algorithms, integration of security gateway functions in Linux/FreeBSD/NetBSD kernels, and testbed design and evaluation. He can be reached at hywei@cis.nctu.edu.tw.

Shao-Tang Yu (styu@cis.nctu.edu.tw) received his M.S. degree in computer and information science at National Chiao Tung University in 2001. His interests include the integration of security gateway functions in Linux kernels, and testbed design. He is now an engineer at D-Link and can be reached at gis88530@cis.nctu.edu.tw.