行政院國家科學委員會專題研究計畫　成果報告

以Linux為基礎的網路安全與頻寬管理閘道器之實作與研究

(I)

計畫類別：個別型計畫
計畫編號：NSC91-2213-E-009-123-
執行期間：91年08月01日至92年07月31日
執行單位：國立交通大學資訊科學學系

計畫主持人：林盈達

報告類型：精簡報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 93 年 2 月 25 日

一，計畫

計畫名稱	以Linux 為基礎的網路安全與頻寬管理閘道器之實做與研究
計畫編號	NSC 91-2213-E-009-123
主持人	林盈達 教授 交通大學資訊科學系
執行機關	交通大學資訊科學系
執行期限	91/08/01–92/07/31

二、關鍵詞

本文關鍵詞—安全閘道器，頻寬管理，防火牆，網路位置轉換，虛擬私人網路，入侵偵測系統，評比，開放程式碼

Keywords—security gateway, bandwidth management, firewall, NAT, VPN, IDS, benchmark, open source

三、中英文摘要

網路安全和頻寬管理對企業來說是重要的議題。這個計畫中展示如何整合開放程式碼，以建造安全閘道器。我們修改核心程式以確保各套件能順利合作。白箱測試顯示 FreeS/WAN 的 3DES 加密是封包處理過程中最吃重工作。而 TIS 則是 requrest/response 處理中最吃重的工作。除了安全功能外，這計畫藉由實現數種 TCP-aware 的頻寬控制方法，來評估頻寬控制方法的優劣。常見的 TCR 方法並不適用於封包容易遺失的 WAN 以及數種作業系統。我們提出的 PostACK 除能保留 TCR 的優點並減少 buffer 的需求達 96%,相對於 TCR 在封包容易遺失的 WAN，可增加 goodput 達 10%。

Network security and bandwidth management have become a critical issue for enterprises. In this work, we first demonstrate how to build a security gateway capable of firewall, virtual private network (VPN), and intrusion detection system (IDS) functions by integrating open source packages. We patch the kernel to ensure interoperability of these packages. Our detailed internal benchmarking reveals that the 3DES encryption in FreeS/WAN tops the ranking of packet processing within kernel for 1518-byte packets, and TIS tops the ranking of request/response processing at the daemon level.

Besides security functions, this work evaluates possible TCP-aware bandwidth control through self-developed implementations in Linux, testbed emulation, and live WAN measurement. The widely deployed TCP rate control (TCR) is found to be more vulnerable to WAN packet losses and less compatible to several TCP sending operating systems. The proposed PostACK approach can preserve TCR's advantages while avoiding TCR's drawbacks. PostACK minimizes buffer requirement up to 96% and has 10% goodput improvement against TCR under lossy WAN.

四、計畫目的

The open source packages we select include Linux kernel [1], ipchains[2], Squid[3], Trust Information System (TIS)[4], FreeS/WAN[5], and Snort[6]. Although each package works well individually, they may not cooperate well to provide specific services. Thus, we trace the packet flows in a gateway to find out the problems, and eliminate those problems by patching kernel and proper setting. Besides security, we proposed and implemented the bandwidth control approach, PostACK, and compared it with the TCR. In addition, three essential packet flows will be discussed in this work. After integrating these packages, we perform a series of internal, i.e. white box, benchmarks. The questions we want to answer include: Who top the processing time of all kernel-space modules and user-space daemon processes, respectively? How much disk and memory does each package consume? How scalable are network address translation (NAT), ipchains, Squid, and TIS? What is the influence of increasing the key length in cryptographic algorithms? Does Snort really examine each packet for suspicious activities? Where are the bottlenecks of these modules?

五、研究方法及結果

I. Selected Packages

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

TABLE 1
PACKAGE INFORMATION.

Package Name	User-space Program	Kernel-space Program	Package Size	Version
ipchains	Management tool	Kernel build-in packet filtering firewall and IP masquerade (MASQ)	63KB	1.3.9
Squid	Daemon (Cache server, transparent proxy, and URL filter)	No	1104KB	2.3
TIS	Daemon (Application proxies, and 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

II. PostACK Bandwidth Control

Each flow should obtain a bandwidth share of $BW_i = BW_c/n$. Recall that in Fig. 1 the RTT consists of Dw_i, the queuing delays at $PCDQ_c$ and $PFAQ_i$, and the neligible round-trip LAN delay. Generally the delay at $PFAQ_i$ approaches zero while the forward-data-packet

queuing delay for TCP is large. Imagine that a Per-Flow Queuing (PFQ) is placed within the class *c* to enforce that each $BW_i = BW_c/n$. Thus, the number of data packets of flow *i* queued before the packet scheduler in Fig.1, $PCDQ_i^{qlen}$, is $min(Wc_i, Wr_i)$- (BDP_i/MSS_i), namely all unacknowledged packets excluding the packets in the WAN pipe. To achieve BW_i, each queued data packet should wait for a period of $(PCDQ_i^{qlen} * MSS_i)/BW_i$. Imagine that the packet scheduler in the forward direction were absent. By delaying each ACK for the same interval $((PCDQ_i^{qlen} * MSS_i)/BW_i)$, the bandwidth of flow *I* will also approach its target bandwidth BW_i. The effects of delaying the data packets in the forward direction by the packet scheduler is identical to delaying the ACKs in the reverse direction since a TCP sender only measures RTT, which consists of bidirectional delays. Gradually increasing the delay of ACKs would not cause Retransmission TimeOuts (RTO) because a TCP sender can adapt the RTO to the newly measured RTTs. In summary, the target bandwidth, BW_i, which keeps only BDP_i/MSS_i packets in the WAN pipe, can be achieved through queuing excessive packets. Either queuing the data packets or the ACKs have the same effects on rate shaping.

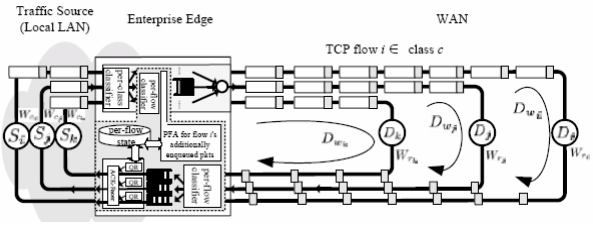

Fig. 1: Efficient PostACK Implementation For Managing Outgoing TCP traffic.

III. Integration

This section discusses three special access types that require extra integration works. Table 2 lists the three access types and their demands of protection. Connections of type 1 access are established from masqueraded private hosts to public Internet servers (except web servers);

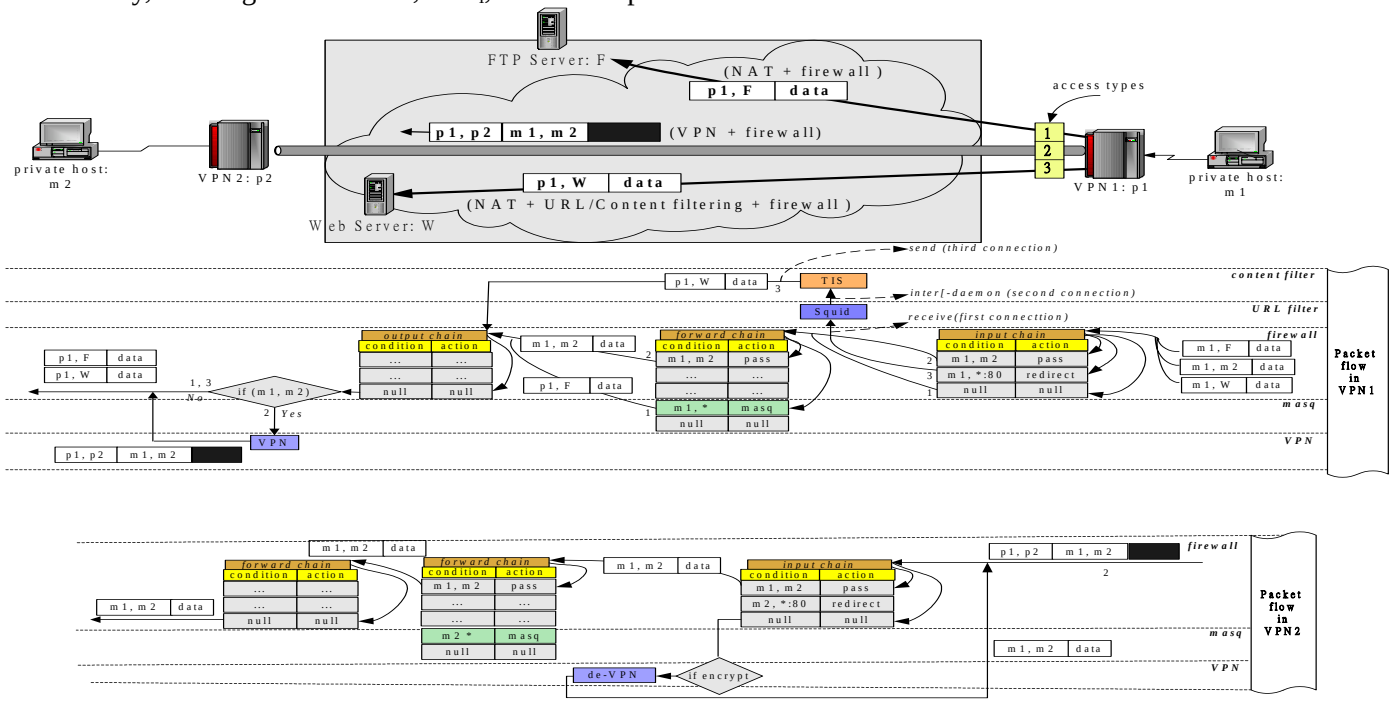

Fig. 2. Packet Flows of Three Access Types.

connections of type 2 are *tunneled* between masqueraded private subnets and do not need URL/content filtering since both sides are trusted regions; connections of type 3 access issue HTTP requests from masqueraded private hosts to web servers to retrieve web pages back to masqueraded private hosts. These accesses should be protected by various firewall actions shown in Table 2. Fig. 2 illustrates the packet flows of the three access types. The upper part is a global view of them, and the lower part details the processing within the two security gateways.

TABLE 2: Access Types

No.	Access Types	Demands of Protection
1	Normal Internet services	NAT, and packet filtering
2	Trusted branch offices communication	VPN, and packet filtering
3	Web services	NAT, packet / URL / content filtering

IV. EXPERIMENTAL RESULTS

We have implemented PostACK and TCR into Linux kernel 2.2.17, together with a practical emulation testbed. The per-flow queuing is achieved by assigning a token bucket policer to each TCP flows. We hereby describe the experimental results. The section investigates the effectiveness of ACK control modules in resolving the unfairness among TCP flows with heterogeneous WAN delays. Test configurations are described in Fig.3. Figure 3(a) demonstrates the classical problem: throughput of a TCP flow is inversely proportional to its RTT. However, when the three flows share a 200KB/s class in a FIFO PCDQ (Fig. 3(b)), the unfairness among the 10ms/ 50ms/ 100 ms flows is alleviated. This is because the RTT measured by flow i (RTT_i) equals to Dw_i+ SUM ($PCDQ_i^{delay}$). The shared PCDQ's queuing delay, SUM ($PCDQ_i^{delay}$), dominates the RTT_i so that the flows are almost fair. Both TCR (Fig. 3(c)) and PostACK (Fig. 3(d)) can further eliminate the little unfairness. Note that these figures are measured at TCP sender side so each peak corresponds to the phase of pumping traffic to the edge gateway. The peaks in PostACK are relatively lower than those in CBQ since whenever a PostACK-applied flow get queued at PCDQ, the QR in PostACK skip the flow's ACK-pacing. So the peak diminishes immediately.

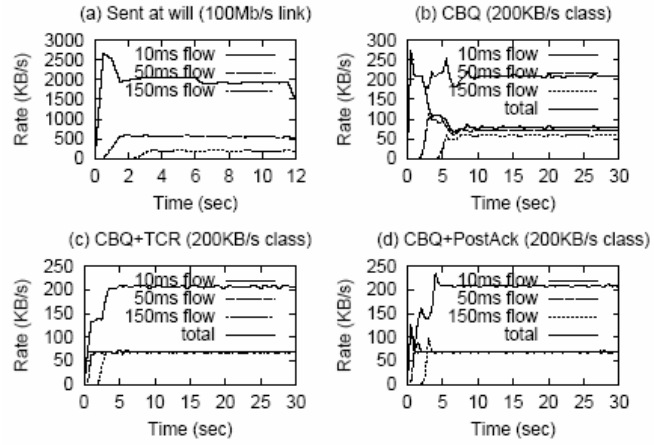

Fig. 3: Fairness among flows in 200KB/s class

V. INTERNAL BENCHMARK

A. 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 3.

TABLE 3
BENCHMAR METHOLOGY.

Category	Benchmark Tools	Benchmark Items
Packet Filter	SmartFlow/SmartBits 2000	Scalability
URL Filiter	self-written HTTP Traffic Generator	Scalability
Content Filter	HTTP Traffic Generator	Scalability
Authentication Algorithms	SmartFlow/SmartBits 2000	Cost of MD5 and SHA1

B. Resources Consumption

1) CPU Consumption: The CPU cost of each kernel module is quantified in Fig. 4 using 64-byte packets. The 0Mbps traffic load is the scenario without background traffic, and 13Mbps is the NLMT of the gateway when enabling all the functions. For a 64-byte packet, the 3DES encryption takes 24.242μs, which is 4 times that of the MD5 authentication, and 12 times that of MASQ. From other experiment results, we observe that the processing time of encryption, authentication, and MASQ depends on the packet size because these modules process the entire packet. Note that the MASQ process re-calculates transport layer checksum. For a 1518-byte packet, the 3DES encryption takes 287.983μs, which is 9 times that of the MD5 authentication, and 31 times that of MASQ. Encrypting 24, i.e. $\left\lceil \frac{1518}{64} \right\rceil$, 64-byte packets requires 581.808μs 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 twenty four 64-byte packets require 24 encryption operations.

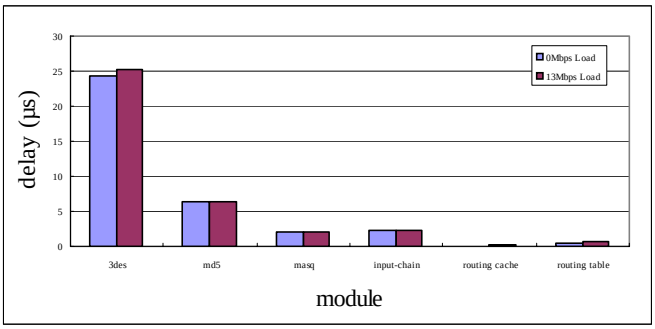

Fig. 4. CPU Cost of Kernel Modules.

Fig. 5 shows the CPU cost of each daemon process. Again, the content filter TIS has some design problems which will be identified later.

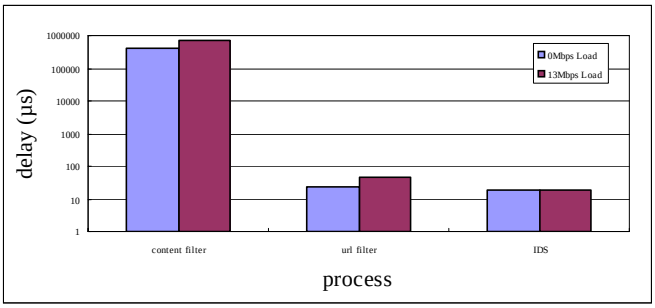

Fig. 5. CPU Cost of Daemon Processes.

2) Memory and Disk Consumption:

Table 4 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 totally 17.3MB disk space and 12.9MB memory mainly due to web caching.

TABLE 4

MEMORY AND DISK CONSUMPTION.

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

C. Scalability Issues

1) Content Filter

As depicted in Fig. 6, the average filtering time for 500K-byte web pages is 68.235ms under 15 concurrent connections, which is not scalable. Further source-code tracing of TIS finds out two implementation problems. First, TIS is found to *fork* a child process to deal with every incomming HTTP request. Moreover, each child process re-reads the configuration file, which involves slow disk accesses. Secondly, TIS performs filtering service with a Finite State Machine (FSM), and TIS reads just one byte of the web page at a time from the socket interface to drive its FSM. This is an inefficient implementation. The worst-case time complexity of TIS is O(*n*) where *n* stands for the size of the retrieved web page. Instead of reading one byte of the web page at a time, reading multiple bytes at a time can reduce *n*.

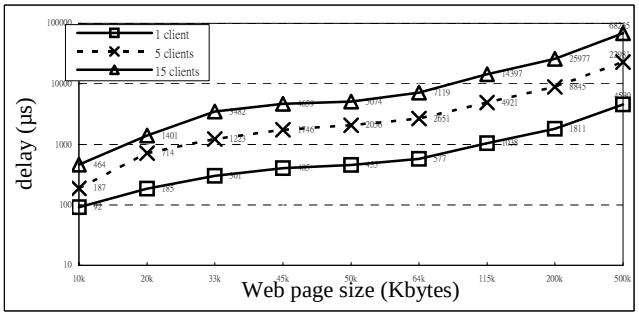

Fig. 6. Scalability of Content Filter in TIS.

2) MD5 and SHA1 Authentication Algorithms

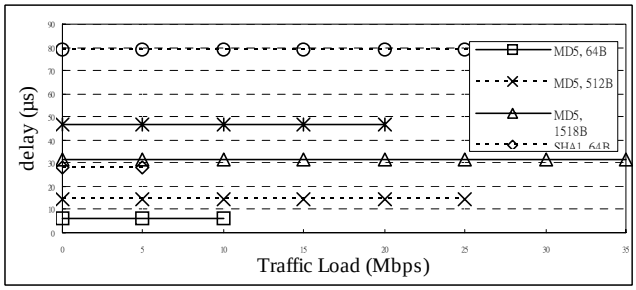

Fig. 7. Cost of MD5 and SHA1 Authentication Algorithms.

This internal benchmark does not include DES and 3DES tests because FreeS/WAN does not support the DES algorithm. Since the MD5 and SHA1 algorithms are derived from the MD4 algorithm, their characteristics are quite similar. The major difference is that the SHA1 digest is 32 bits longer than the MD5 digest. Thus, SHA1 executes slower than MD5 on the same hardware as shown in Fig. 7. The processing time of 1518-byte packets is 31.89µs and 79.84µs for MD5 and SHA1, respectively. The digest generation process includes (1) append padding bits to the original message; (2) append the length of the original message; (3) initialize input key; (4) process the message in a sequence of 512-bit blocks; (5) generate the output digest. Therefore, as the packet size increases, the time for digest generation gets longer. The worst-case time complexity of these two authentication algorithms are O(*n***m*) where *m*, and *n* stand for the key length, and the packet size, respectively.

六、結論

This paper provides the experiences of integrating many open source packages into a security gateway. Besides, a PostACK TCP-aware bandwidth control approach is proposed. PostACK minimizes buffer requirement up to 96% and has 10% goodput improvement against TCR under lossy WAN. In internal benchmark, we examine the CPU/ memory/ disk consumption of the open source solution, and investigate the scalability of each key module. Finally, observations of benchmarking and suggestions for performance improvements are presented here. Table 5 summarizes the observations of our benchmarking. The results of our study reveal that ipchains and FreeS/WAN are viable compared to commercial products, but TIS and Snort have performance problems.

TABLE 5
SUMMARY OF OBSERVATIONS.

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& 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
TIS	CPU-intensive	Increasing the number of HTTP connections and the size of the retrieved web page	1. Parse configuration file for each request 2. Only read one byte of the web page from the socket interface at a time	O(*n*); *n*: size of the retrieved web page
Masquerade	CPU-intensive	Increasing the number of private-to-public connections	Data structure of masquerade table	O(*n*); *n*: number of private-to-public connections
FreeS/WAN	CPU-intensive	Using the stronger algorithms	Too many computation 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

七、參考文獻

[1] Linux kernel, http://www.kernel.org .
[2] ipchains, http://netfilter.filewatcher.org/ipchains/ .
[3] Squid, http://www.squid-cache.org .
[4] TIS, http://www.tis.com .
[5] FreeS/WAN, http://www.freeswan.org .
[6] Snort, http://www.snort.org .