第 4 部

ネットワークトラフィック統計情報の収集と解析

第 1 章

MAWI ワーキンググループ

MAWI(Measurement and Analysis on the WIDE Internet) ワーキンググループは、トラフィックデータの収集と解析、また、データの保存と利用に関する活動を行なっている。

回線利用率やプロトコルの分布を示すマクロなトラフィックデータは、故障や問題の検出を容易にし、バックボーントラフィックの傾向の分析に役立つと同時に、ネットワーク機器選定や将来のバックボーンの構成、容量設計に欠かせない重要なデータである。

一方、パケットトレースに代表されるミクロなトラフィックデータは、生の記録をもとにさまざまな情報を抽出することが可能であるため、プロトコル設計、トラフィック理論をはじめ、あらゆるネットワーク研究にとって貴重なデータとなる。その反面、生データにはユーザのプライバシ情報も含まれるため、取り扱いには十分な注意が必要とされる。

現在、MAWI ワーキンググループでは、パケットトレースをより有効に利用するため、プライバシ情報をスクリーニングした、安全に取り扱えるトレースデータの作成を実験中である。また、これと並行して、複数のサンプリングポイントでの定期的なトレースデータの収集、収集したデータをアーカイブするサーバの設置、収集データを自動処理するツールの開発等を進めている。

今回の報告書では、まず、第 2 章で従来から継続的に行なっている国際線トラフィックの統計データを示す。

第 3 章では、(株) KDD 研究所との共同研究として行なっている、WIDE 国際線のパケットトレース収集について報告する。本研究では、パケットトレースに求められる情報とユーザプライバシーの保護について考察し、1999 年 4 月現在稼働中の実験システムの構成と実装の詳細、今後の課題について論じる。

第 4 章では、パケットトレースをもとにウェブサーバの性能解析を行なう手法について報告する。ウェブサーバの性能評価には、アクセスログをもとにした解析やベンチマークを用いた評価が一般的であるが、ここではそれらの問題点を上げて、パケットトレースから実働中のサーバ性能の評価を実現する手法を提案している。本手法を用いて、1998 年夏の甲子園大会のウェブサーバの評価を行ない、ログからの解析と比較した結果と、実験室環境において異なる CPU 性能のサーバを評価した比較結果を示す。

第 2 章

WIDE 国際線のトラフィック

2.1 プロトコル別トラフィック

国際線のトラフィックは、WIDE 藤沢 NOC と WIDE サンフランシスコ NOC 間の 1.5Mbps の国際回線上を通るトラフィックを NNStat を用いて計測した。

表 2.1は、各 IP プロトコルについて、月毎のトラフィック量を 1 日当たりで平均したものである（単位：キロバイト）。各月ごとに、IN の行は国外から国内に向かうトラフィック、OUT の行は国内から国外に向かうトラフィックをそれぞれ表している。回線利用率は、1.5Mbps の国際回線を用いて 1 日に送信できる総トラフィック量に対する、各月の 1 日平均のトラフィック量の割合を求めることにより算出した。図 2.1、図 2.2は、表 2.1をもとに、国外から国内に向かう平均トラフィック量の推移、国内から国外に向かう平均トラフィック量の推移をそれぞれグラフ化したものである。

過去 6ヵ月間、月の平均トラフィックが双方向とも頭打ちとなっている、また回線利用率をみると、国外から国内に向かう方向では、最も利用率が大きい月は 2 月で 95.80%、国内から国外に向かう方向では、最も利用率が大きい月は 1 月で 52.52%となっている。利用率の算出に用いたのが 1 日平均のトラフィックであることを考えると、非常に高い利用率であるとみなすことができるとともに、定常的に混雑した状況であるといえる。

プロトコル別では、国外から国内に向かう方向では TCP が 94%程度、UDP が 5%程度、ICMP が 0.5%となっている。国内から国外へ向かう方向では、TCP が 80%程度、UDP が 14%前後、ICMP が 3%となっている。

2.2 TCP ポート別トラフィック

表 2.2は、TCP ポート別の 1 日あたりのトラフィック量の推移をキロバイト単位で表したものである。各アプリケーションによるトラフィックは、HTTP によるトラフィックに圧迫され、それぞれの割合は漸減している。その傾向は、国外から国内向きのトラフィックで顕著であり、HTTP の割合は 1998 年 10 月以降 89.79%から 1999 年 2 月には 92.37%まで増加しており、ほとんどすべて HTTP と言えるような状況である。HTTP 以外では各アプリケーションの占める割合は大きく変化しているわけではない。国内から国外向きのト

表 2.1: 国際線のプロトコル別トラフィック

月		ICMP	IGMP	IP/IP	TCP	UDP	Other	合計	回線利用率
10	IN	329,951	25	2,683	12,463,100	498,962	2,225	13,296,945	80.16%
	OUT	135,943	3,181	220,332	7,357,451	991,102	4,860	8,712,869	52.52%
11	IN	136,309	8	1,066	13,974,592	531,916	3,468	14,647,359	88.30%
	OUT	228,910	3,551	32,121	4,340,201	684,084	7,445	5,296,314	31.93%
12	IN	128,518	37	1,655	13,175,282	909,425	952	14,215,870	85.70%
	OUT	233,012	639	362	4,542,933	628,546	6,897	5,412,390	32.63%
1	IN	94,698	9	1,120	12,603,517	877,611	2,817	13,579,772	81.86%
	OUT	229,221	584	4	5,228,291	1,128,247	10,478	6,596,826	39.77%
2	IN	90,185	11	643	14,794,601	1,003,207	3,007	15,891,653	95.80%
	OUT	98,428	585	74	5,172,307	1,212,621	12,180	6,496,194	39.16%
3	IN	166,186	51	394	14,113,927	611,115	2,924	14,894,596	89.79%
	OUT	59,502	502	640	4,700,636	775,790	11,422	5,548,492	33.45%

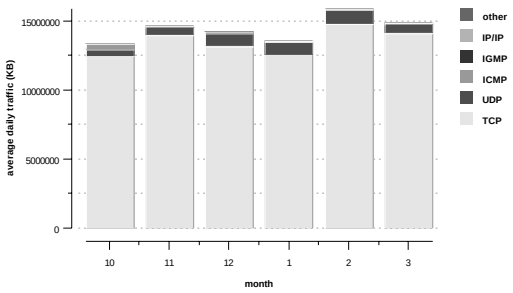

図 2.1: 国外から国内向けのプロトコル別トラフィック量推移 (1 日平均)

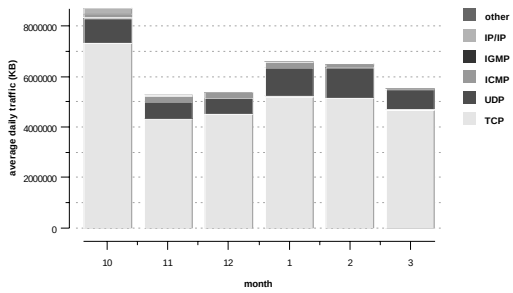

図 2.2: 国内から国外向けのプロトコル別トラフィック量推移 (1 日平均)

ラフィックは逆の方向と比較して、その 3 割程度の量となっている。そのうち、SMTP が安定したトラフィック量となっているのに対して FTP-Data は漸減傾向にある。

図 2.3、図 2.4は、表 2.2をグラフ化したものである。HTTP のトラフィックがかなりの高い割合を占めていることがわかる。言い替えると他のトラフィックは極めて少なく、またはないに等しいような状態であると言える。過去の報告書で問題とされていた Others に分類されるトラフィックは 0.4%前後と極めて少ない。

2.3　UDP ポート別トラフィック

表 2.3は、UDP ポート別の 1 日あたりのトラフィック量の推移をキロバイト単位で表したものである。また、図 2.5、図 2.6はそれをグラフ化したものである。

表 2.2: TCP ポート別トラフィック

		HTTP	FTP-Data	SMTP	NNTP	FTP	Telnet	BGP	DNS	Other
10	IN	8,932,501	583,529	379,825	13,671	11,099	4,421	4,110	11,255	7,423
	OUT	4,144,762	412,762	590,093	14,219	13,902	8,386	4,568	10,953	19,176
11	IN	10,834,400	644,581	418,947	21,699	18,000	3,005	4,174	3,659	7,713
	OUT	2,456,969	194,010	497,701	11,057	7,419	4,093	4,499	10,380	15,233
12	IN	10,395,740	645,392	453,677	24,084	24,381	6,835	4,272	7,272	6,362
	OUT	2,622,526	433,333	494,013	10,144	9,176	4,573	4,935	10,652	17,773
1	IN	9,878,288	579,585	347,272	22,955	24,129	3,162	4,177	2,380	6,621
	OUT	3,159,791	379,829	449,655	9,794	7,665	6,750	4,681	12,331	16,315
2	IN	12,587,621	380,992	417,261	16,892	27,825	3,854	4,019	1,170	6,040
	OUT	3,010,192	253,665	502,570	7,853	8,366	6,214	4,077	13,420	17,800
3	IN	11,661,245	501,315	398,994	20,975	27,515	4,234	4,291	1,030	5,221
	OUT	3,208,628	142,337	528,608	7,277	8,977	4,525	4,721	12,925	16,339

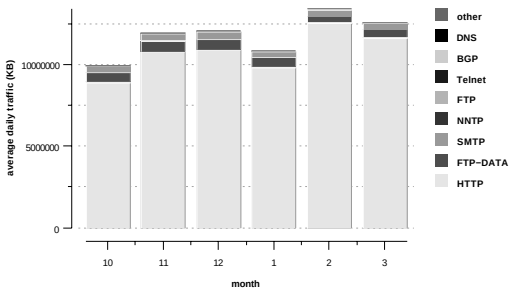

図 2.3: 国外から国内向けの TCP ポート別トラフィック量推移 (1 日平均)

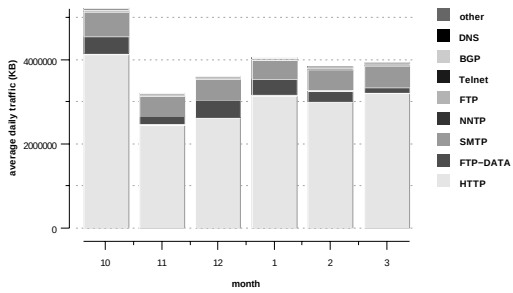

図 2.4: 国内から国外向けの TCP ポート別トラフィック量推移 (1 日平均)

どちらの方向に関しても DNS によるトラフィックが相変わらず多く、97%前後までを占めるようになっている。このような状況では、TCP の場合と同様に、Other に分類されるトラフィックも他のアプリケーションのトラフィックと同様に極めて少ない割合を示すようになってきている。

2.4 まとめ

WIDE の国際線である藤沢-サンフランシスコ線は極めて高い回線利用率となっており、結果として、TCP・UDP ともに HTTP および DNS といった特定のアプリケーションのトラフィックが他の追随を許さずむしろ圧迫しているような状況になってきている。

これから行われる国際回線の増強に伴う高速化・広帯域化によって、トラフィックの状

表 2.3: UDP ポート別トラフィック

		DNS	AFS	NTP	SNMP	Other
10	IN	232,522	39,307	488	105	63,292
	OUT	829,757	12,660	3,007	2,886	24,795
11	IN	345,010	25,571	454	68	3,687
	OUT	548,368	14,274	1,404	177	9,042
12	IN	766,316	23,781	756	90	4,804
	OUT	544,096	7,602	1,642	2,016	9,710
1	IN	722,753	15,015	577	201	10,145
	OUT	1,044,950	4,320	1,654	2,714	10,559
2	IN	763,869	9,239	435	240	8,732
	OUT	1,015,137	4,008	1,533	288	9,296
3	IN	479,947	6,875	551	169	2,873
	OUT	641,749	3,194	1,754	138	12,575

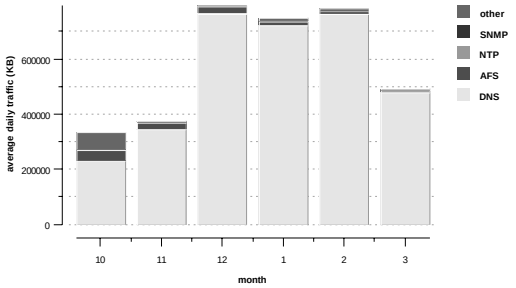

図 2.5: 国外から国内向けの UDP ポート別トラフィック量推移 (1 日平均)

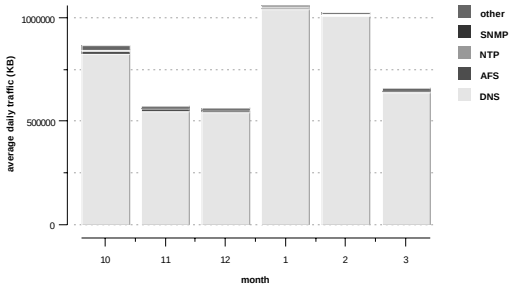

図 2.6: 国内から国外向けの UDP ポート別トラフィック量推移 (1 日平均)

態がどのように変化するかを注目して分析することが必要である。

また回線の有効利用という点から、ICMP や DNS などトラフィックに多く含まれる crack と思われる無駄なトラフィックをバックボーン上から排除していくことも重要である。特定の IP アドレスからのトラフィックが全体の数%にも及ぶという状況 (通常のトラフィックは全体の 0.1%にも満たない通信が大部分) が日々見られる。今後はこれらを対象とした分析も必要となる。

これまで使用している NNStat は歴史の長いトラフィック収集のアプリケーションソフトウェアであるが、収集の設定に自由度、柔軟性が高く、未だに十分な基本機能を有している。今後のトラフィック収集解析においても NNStat の長所・短所を検討し改良しつつ利用して行きたいと考えている。

第 3 章

An Internet Traffic Data Repository

3.1 Introduction

Currently, many issues regarding the quality of services or Internet traffic models such as RSVP [27, 133], DiffServ [23, 95], are under evaluation. However, it is difficult to verify these traffic models and service models based on the actual network traffic data which is commonly accessible and to analyze these models with sufficient detail. After taking into account the current issues involved in traffic data capturing, this paper suggests a method for supplying actual traffic data in order to evaluate these models.

The first commonly used traffic data were captured on October 3rd, 1989 at the Bellcore Morristown Research and Engineering Facility [79]. The data files were in ASCII format, consisting of one 20 byte line for each arriving Ethernet packet. Each of these lines contains a floating point time stamp and an integer length representing the Ethernet data length in bytes. The time expressed to six places after the decimal point was intended to the appearance of microsecond resolution. Timestamps however, are only accurate in milliseconds due to the actual resolution (4 microseconds) of the hardware clock, jitter in the inner code loop, and bus contention.

The next generation of traffic data collections were based on the *tcpdump* program. Here, V. Paxson and S. Floy gathered two to four hours of such traffic at Digital's primary Internet access point on March 1995. The raw traces were made using *tcpdump* on a DEC Alpha running Digital's OSF/1 operating system which includes a kernel filter with capabilities similar to those of BPF. Sanitize script was used which renumber hosts and strip out packet contents in order to address security and privacy concerns. Timestamps have millisecond precision even though they are reported in figures using six digits past the decimal point [103]. The several independent data collections have been made based on almost the same method. Some of these data addressed specific protocols such as HTTP.

The ever increasing amount of Internet traffic is forcing each Internet service provider or ISP to analyze whether or not their network resources are sufficient to maintain customers' satisfaction. The previous traffic data addressed specific protocols or applications and

did not reflect all traffic through each ISP. The data should however, give us information on any application or protocol at any time. The need to capture all packets also arises from requirements of other approaches such as an academic research community studying mathematical models of Internet traffic.

We introduce a new method to create a database which contains actual traffic data holding sufficient amounts of information to allow analysis of end-to-end characteristics of Internet or intranet traffic without exposing the users' privacy. The required data to be captured will be described as well as the format of the resulting database.

3.2 Traffic Data Repository

Under the current operation environments for the Internet, monitoring at intermediate nodes is usually done by SNMP [30] related query tools to see a simple set of metrics, such as utilization rates, packet discard rates, etc. for transmission links. This kind of parameters have an advantage to be monitored without any stress on the networks. The traffic data repository, which is also monitored without any stress on the networks, is intended to provide the operators with more information on the networks to investigate their services.

3.2.1 Requirements for Traffic Data Repository

Currently accessible tools and data possess several inherent problems associated with modeling and simulations of Internet or intranet traffic. Publicly available tools and public databases with sufficient information based on impartial evaluation measure might serve to encourage this type of research.

There are two issues to be considered in traffic data repository. The first problem is the measuring locations. The current fields of research, such as management of quality of service on the Internet, require end-to-end flow analysis. This means that measuring at an intermediate node will not be sufficient for analyzing end-to-end traffic. Although the currently used intermediate monitoring node is inadequate, it is still the only way for an Internet service provider to determine service quality and therefore an essential task.

The second problem is to what extent to make measurements at intermediate nodes in order to monitor or study the traffic later on with as much information as possible. Other factors that must be considered are the time precision, amount of data to be recorded for each packet as well as hiding private information from the measured traffic data before making this data available for public use.

After all, the following requirements must be satisfied:

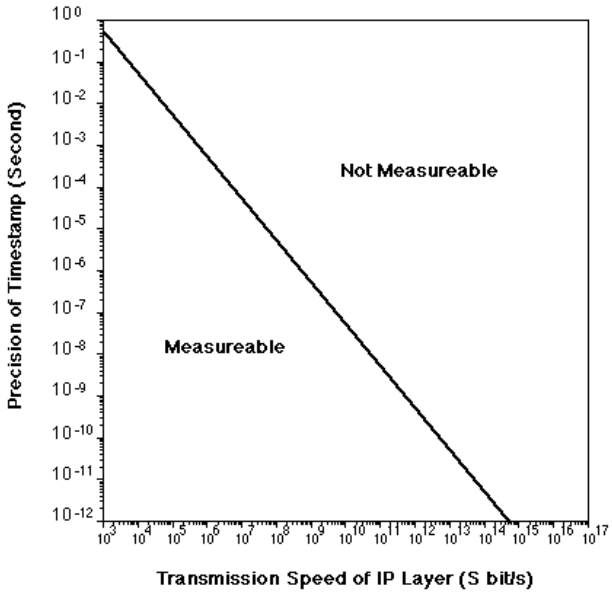

図 3.1: Required Timestamp Precision vs. Packet Transmission Speed of IP Layer

1. All packets must contain as much information as possible.
2. User privacy must be assured.
3. Timestamps must have adequate precision.

Time precision is a critical for performing analysis in the future on the Internet. The current speed of the major Internet backbones is increasing from 156Mbit/s or 622Mbit/s to 2.4Gbit/s. It is further assumed that a backbone with a speed in tera bits per second will be attained within a few years. Since the shortest IP packet is an IGMP packet of 28 bytes long, the minimum timestamp precision must be equal to the resolution to distinguish this packet from others. This resolution is on the order of 28*8/S where S stands for the packet transmission speed in bit/s over the IP network layer. Figure 3.1 shows the current millisecond precision of timestamps will only be adequate for 100kbit/s IP transmission lines. Future tera bit per second line monitoring will require 3.68E-10 seconds of precision.

3.2.2 Modes of Traffic Data Capturing

We categorize traffic capturing into two methods or modes, namely *stateless* and *statefull*. Stateless capturing means a record of a packet does not depend on the previously captured traffic data or packets. Processing to make a traffic data is only done over the information provided by the current packet. Most of the currently available tools, such as *tcpdump*, are

based on this packet-by-packet scheme. Statefull capturing on the other hand records the information of a packet based on the previously captured packet records. Every capturing based on identifying flows is considered as a statefull capturing. Statefull capturing will be required to compress the captured data with TCP header compression [71], RTP header compression [31], etc.

Two advantages are known for stateless capturing. Firstly it is simple and does not require a heavy processing overhead on a capturing machine. Secondly there is no need to predetermine what higher layer protocols are running through the observed node since capturing information is uniform for every IP packet. A disadvantage is that it is theoretically necessary to record entire packets to analyze the traffic. This requires a large amount of disk space and also a high speed recording facility for capturing to follow the link speed which is assumed to be Gbit/s or higher.

On the other hand, statefull capturing is based on the flow analysis to decide the recording part of the IP payload and to compress the amount of data. The advantage is that it requires less disk space and a moderate speed of recording facility, with the introduction of a high processing overhead for the flow identification, tracing state transitions of higher layer protocols, data compression, etc. In addition to these disadvantages, it is assumed to have a prerequisite knowledge of the captured protocols. Since the evolution of the Internet protocols and applications is very fast, it is very difficult to predetermine all the protocols used in the traffic through the observed node.

3.3 Data Format

Our current implementation is based on stateless approach. Two data formats to record traffic traces are supported in the system. One is a 24byte fixed format for a captured packet as shown in Figure 3.2. This includes timestamp in 64bit attached by the BPF driver followed by IP source and destination address in 32bit each, total length in 16bit, protocol in 8bit copied from the IP header. Other fields in the IP header are just ignored. There are one flag byte and two 16bit fields as well. The flags byte is copied from TCP flags when appropriate. The two 16bit fields are used to represent source and destination port number for TCP or UDP, and type and code for ICMP. Some types of ICMP messages include IP headers of target IP datagrams as well as first 64bits of the payloads In this case, another 24byte record is used to represent the target IP datagram with clearing its timestamp field. This format is intended to make the record *small*. So monitoring on a higher speed network would be possible.

The other format supported in the system is shown in Figure 3.3, which is compatible to *PCAP* library [86]. This format is not so compact, however, several well-known tools such

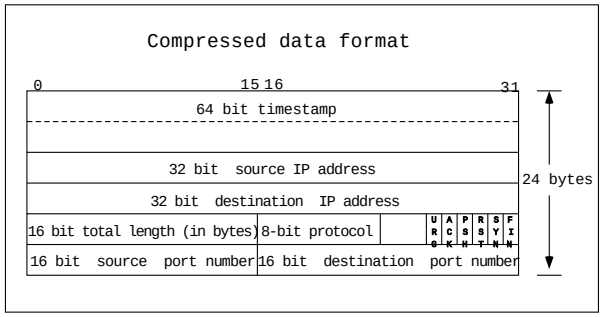

図 3.2: Captured Data in Compressed Format

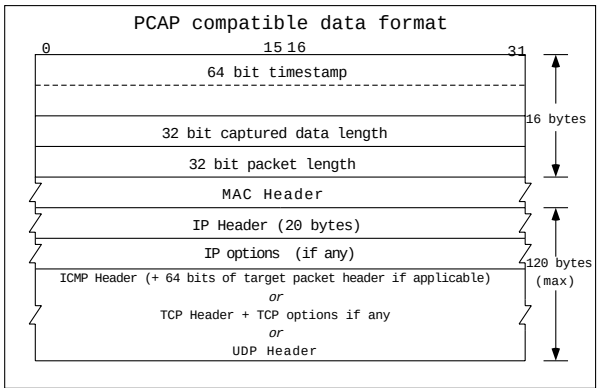

図 3.3: Captured Data in PCAP Format

as *tcpdump* can be used with the format. To save the amount of data and not to record user data, only IP headers (with options if any), TCP headers (with options if any), UDP headers, and ICMP header (with initial 64bit of target datagram payload where appricable) are recorded. Each record of the format is not of the fixed length. As TCP options are recorded with this format, it is suitable for TCP oriented analysis.

As for UDP packets, RTP is now popular to be used over UDP. RTP packets have data reflecting end-to-end throughputs for UDP applications, and RTP as well as TCP packets are very important to capture to understand end-to-end service quality. Since it is very difficult to identify an RTP packet based on the UDP payload of the corresponding packet, Statefull capturing is required to identify RTP packets. When specified in an option, the program records the first 20bytes of UDP payload for this purpose.

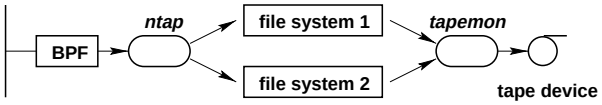

図 3.4: Structure of Capturing Software

3.4 Software Structure

The paper then considers the architecture and the design policy of a new traffic data repository tool to solve the above problems, describing what parts of IP headers and data must captured, what timestamp required for what data, etc.

A data compression method for these captured data is also investigated to reduce the total disk space and to increase the capture speed. The end user's privacy resides in IP address and port number. This paper further presents an effective address scrambler based on the network interface and the address classes and also shows a naive address scrambling scheme having a security hole.

As the system intends to capture traffic data over long period with limited storage resources, it writes captured data to the file system while background process copies the file to the cartridge tape. Providing the change operation of the tape media, with a mechanical autochanger or with timely operator assistance, the system can capture the traffic data for any period, virtually.

The system consist of two processes as shown in Figure 3.4. *ntap* process captures the traffic through a BPF, converts the data to an appropriate format, and writes the data to the file. *ntap* is configured with a list of file system. When *ntap* changes the file to be output periodically, for example, once an hour, it checks the remaining space of the current file system. If necessary, it changes next available file system.

Another process *tapemon* checks each file system periodically. When the growth of file system utilization stops, *tapemon* assumes that *ntap* program writes to another file system. After a few minutes, *tapemon* dumps the files in the non-working file system to the tape. Before the tape is exhausted, *tapemon* rewinds the tape and send a notification to the operator via email.

The software runs on BSD/OS operating system version 3.1 and 4.0. As the system dependent feature the system assumes is only BPF interface, the software can be easily ported to other BSD flavored Unix. In order to avoid access collisions, distribution of SCSI attached devices (hard disks and tape drives) to different SCSI buses is important. In order to eliminate BPF buffer overruns, it is necessary to make BPF buffer size larger. The default maximum size of BPF buffer size in BSD/OS was 32kB. This number has been

compiled into the BSD/OS kernel and recompilation of the kernel after changing this into at least 256kB is necessary.

As the Unix operating system is not a realtime operating system, non-blocking *close()* system call is not supported. We observed BPF buffer overruns may happen when we switched the output file. This is because there may be large amount of un-flushed data in the kernel buffer and *close()* requires flush the data first. A technique to avoid this is to delay file close operation for a few minutes so that *update* daemon may perform *sync()* system call to flush the data to the disk asynchronously.

When the captured data is exported, we need to encrypt any field related to the users' privacy. When we randomizing the IP addresses in the captured data, following rules may apply:

- Need to keep one-to-one mapping, i.e., once a address *addr1* is mapped to *addr2*, any occurrence of *addr1* in the raw captured data should be mapped to *addr2*.
- Multicast addresses are mapped to multicast addresses.
- Private addresses, Class E addresses, and loopback address are not mapped.
- Use a random number generator to map the addresses.
- Checksum fields in TCP or UDP headers are cleared.
- IP headers in ICMP payload are also a subject of address mapping. Furthermore, the initial 64bit of the target datagram of an ICMP message should be removed if it is not TCP header nor UDP header.

3.5 Experimental Capturing

The rest of the paper is devoted to issues regarding implementation of this repository tool for BSD UNIX. The data format of captured traffic is also described. Field experiments with this tool were performed at a network operation center of WIDE, a Japanese academic network, for two months. As the WIDE Internet has two international links, we run a capturing system for each international link.

The configurations of the capturing systems are shown in Figure 3.5. The gateway system to the international link consists of two routers connected by a shared ethernet. A capturing system is attached to the ethernet to monitor the international traffic. A system for a T1 link is equipped with a Pentium-II 333MHz, two 9GB WIDE-SCSI disks, and a DDS3 (capacity is about 15GB for the captured data) tape drive on another SCSI bus.

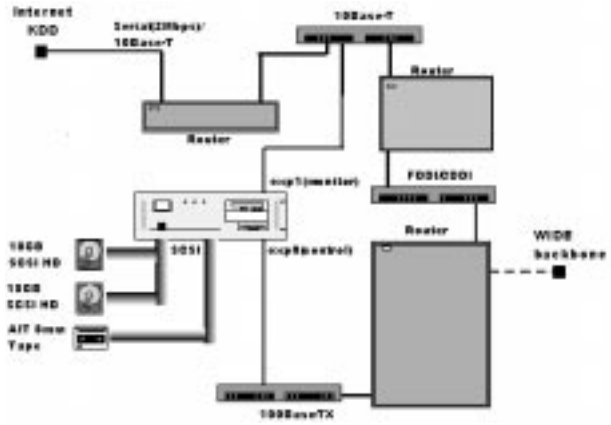

図 3.5: Configuration of Experimental Network for Traffic Capturing

For another link of 1.9Mbit/s bandwidth, a system with a Pentium-II 400MHz, two 18GB WIDE-SCSI disks, and an AIT drive (capacity is about 45GB) is assigned.

In order to check the ability of the system, a similar system with single 9GB WIDE-SCSI disk and a FDDI interface is tested on a backbone FDDI network of the University of Tokyo. While the peak traffic exceeds 20Mbps, the system reports no dropped packet at BPF layer over more than 10 days with an even worse condition where *ntap* and *tapemon* access file systems on a single disk.

The cumulative distribution of the packet length for the data captured from 11:00:00 to 12:00:00 on Feb. 22, 1999 is shown in Figure 3.6. According to this data, the average of the packet length is 361 byte long, and the top two lengths, namely 40 and 1500 byte, are observed to occupy 41% and 14% of the total number of packets. Packets of 1500 byte long are notable in our traces. This may reflect a popular usage of the applications like Web or FTP in the current Internet. Of course, the characteristics of our traces are very different from the Bellcore' results [79], but further investigations will be necessary to compare ours with other traces.

In the middle of this experiment, the bandwidth of the latter international link was increased from 1.9Mbps to 10Mbps. Hence the resulting data contain a history of the network behavior from a heavily congested state to a normal state. Some data analyses of the captured traffic are also presented.

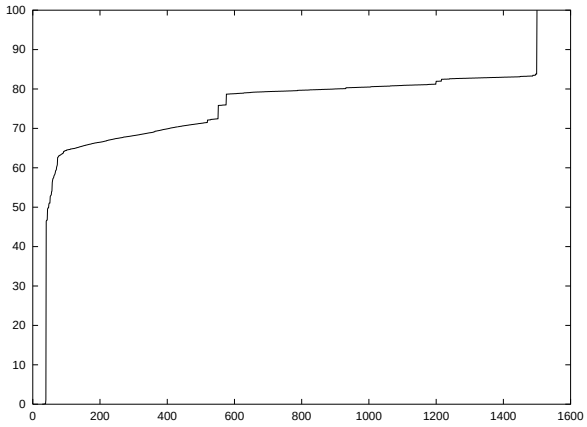

図 3.6: Cumulative Distribution of the Packet Length for the Captured Data

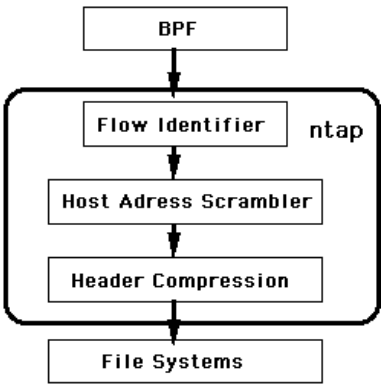

図 3.7: Extension of ntap to State-full Capturing

3.6 Discussion and Future Directions

Although the current version is a state-less capturing system, we present a block diagam of our future state-full version in Figure 3.7 to show that state-full capturing for a giga bit network will be possible without any disk access bottleneck, using a conventional SCSI disk drive. The three modules, flow identifier, host address scrambler and header compression need not to be a single process nor run on a single processor.

The first module, flow identifier, must follow a state diagram of each protocol to identify each flow. At a gateway router, a number of flows will exist to be processed and some of them may be terminated abnormally. The latter will increase the vertual number of flows to be processed. These cause a task of flow identification to be expensive.

After the flow identification, host address scrambling is applied to each packet, which renumber hosts in order to address security and privacy concerns. This scrambling can be eliminated if the operator wants to do his job with the actual host addresses.

Then at the next stage after making a secure IP header, the TCP [71] and the RTP header compression [31] can be applied to all the TCP and RTP packets, which reduces 40 bytes of IP and TCP header into 3 bytes, and also 40 bytes of IP, UDP and RTP header into 2-4 bytes in average. This means only 8 bytes of time stamp, 2 bytes of flow ID, 2 bytes of data length, and 2-4 bytes of header data will be enough to record for each packet. A record of 14-16 bytes for each packet with an average packet length of 361 bytes, solves the disk access bottleneck.

Assuming 90% of the Internet traffic is TCP based applications, it may be possible to capture IP packets through a datalink up to 2.3Gbps with disks attached to Ultra2 SCSI or Wide Untra2 SCSI, assuming that the average sustained transfer rate is around 12MB/s. Thus the disk access speed is not the bottleneck of capturing packets on gigabit networks. There is, however, a significant processing overhead for flow identification and header compression . Specialized hardware devices or parallel processors will be required for this purpose.

All output lines are preceded by a timestamp. The 64 bit timestamp means that the maximum observation period will be 213.5 days, measured from the start of capturing, even if the time precision of one bit is as small as 1.0E-12 second. This is sufficient for monitoring IP traffic over future links at tera bit/s speed. There is, however, a limitation on our current implementation. Firstly, the timestamps attached by the BPF driver in the UNIX kernel is of *struct timeval*. This time value is not started from the beginning of the capturing but is measured between Jan. 1, 1970 and 2038 in the precision of 1 micro second. Although the clock data returned from this function is 64 bit long, the timestamp for the current clock time is as accurate as the kernel's clock, that is the order of micro second. Further more the timestamp reflects the time the kernel first saw the packet. There is some time lag between when the network interface received the packet from the wire and when the kernel serviced the 'new packet' interrupt. Because of these factors, the accuracy of timestamp is restricted to the order of several milliseconds.

To overcome this problem, specially designed network interfaces must also be developed for timestamps. These interfaces must have a function to attach by themselves a timestamp to each received packet or frame with the nano or pico second order of precision counted from the beginning of capturing.

A notifying usage of the resulting traces is to estimate the amount of the exchanged information through the target node. There have been many evidences on the asymmetric traffic between the US and Japan, based on the inbound and outbound packet data on the gateway routers, but there has not been a further detailed analysis on this fact. Assuming

the side to initiate the connection receives the valuable information, which is usually the case for TCP based applications like FTP, TELNET, Web accesses, etc., we can estimate what amount of information is transferred to which direction during the observed period. The procedure is very simple. We just have to identify which side first sends a TCP packet with a SYN flag.

3.7 Conclusion

We present a method to create a database containing real traffic data, which has sufficient information to analyze the end-to-end characteristics of Internet or intranet traffic without disclosing private end user information. Our current version of a program running on UNIX implements these features. As we mentioned in the earlier discussions, some improvements for capturing performance and functionality will be required for our program to be used for links faster than our experimental links. Although far from an ideal tool in terms of capture speed, it has been used to make traces of WIDE Project traffic since January 23, 1999 at Internet gates. All the processed traces are intended to be public and will be uploaded to our FTP server after completing the current experiment.

第 4 章

A WWW Server Performance Measurement System

4.1 Introduction

In the operation of WWW services, the quality of service is now a first priority. Administrators of WWW services have to know how their services are working, and try to maintain their services to fulfill the service subscribers' requests as much as possible. In order to improve the quality of the WWW servers technically, their operators want to aware any changes on the performance indices such as the average and peak numbers of HTTP [53] requests rate, the storage usages of both physical memory and disks, and the amount of total traffic for the services. Through watching these performance indices, the operators can decide the way of improvement on WWW servers. However, measuring the performance of the WWW servers is quite hard because of several reasons;

- The WWW servers are ordinarily implemented as application-level programs running on several computer platforms so that many factors can make influences on the server performance.
- The WWW servers that the operators want to know their performance are likely to be working as a "running" system, so that suspending their service only for making benchmark or other performance measurement is almost impossible.
- There are few tools and utilities that can be used for performance evaluation and tuning of the WWW servers.

As the result, the administrators have to manage their servers only with their intuitions and experiences. This may cause several difficulties on the performance tuning. Therefore, the administrators are willing to have several performance indices how the WWW server sufficiently processes users' requests.

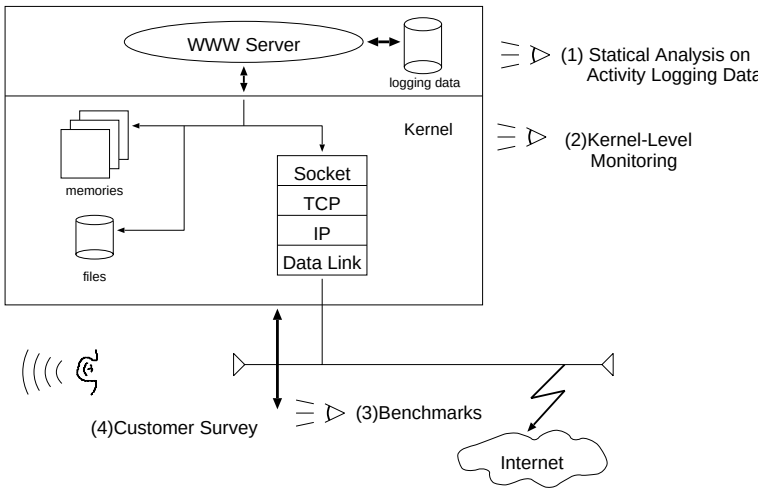

図 4.1: Several ways to measure the performance of WWW servers

The goal of our work discussed in this article is to develop a powerful performance measurement tool called "ENMA" for the WWW servers. As mentioned above, the performance measurements through the benchmark or any mechanisms installed inside the server itself are not practical for applying them to "running" WWW servers. Our approach is, therefore, to make performance measurement through the packet monitoring. Its fundamental idea is to observe the behavior of WWW services from outside, and estimate several performance indices.

4.2 How Can We Know the WWW Performance?

Several ways to measure the performance of WWW servers have been developed so far: (1) log data analysis, (2) kernel level monitoring, (3) benchmarks, and (4) customer survey (see Figure 4.1).

In the method (1), the statistical analysis is applied on the activity logging by WWW servers. This is a popular way to know the performance of WWW servers. Almost all of the servers can generate various types of activity logging data as their log files. The statistical analysis of these log files allows us to know the behavior of WWW servers such as the number of accesses, clients' access patterns, the request processing time, etc. However, this kind of analysis cannot provide any hints on behaviors that the WWW server invokes inside the OS kernel such as its memory consumption, the number of disk I/O, the frequency of the network accesses, etc.

To solve this issue in the method (1), the kernel level monitoring, as the method (2), is

frequently used. One of the popular ways in many UNIX operating systems is to use a utility *ktrace* to monitor the WWW server. With this utility, most of the kernel level behavior including system calls, I/O, signal processing, and file system operations can be revealed. With the other utilities such as kernel debugging utilities and a kernel-level process profiler can be used also for this purpose. However, this method has serious drawbacks: its disk storage consumption and performance interferences. These kernel-level monitoring utilities generate a large volume of files in which kernel-level monitoring data are stored. Therefore, this method can be applied only for the period short enough the disk storage does not run out. Even if the huge disks are prepared, still the performance interferences caused by these utilities cannot be negligible. Since these utilities invokes a large number of disk I/O for making a log recorded at each system call, therefore, the performance degradation is quite large. In order to minimize the performance interferences, monitoring the kernel-level behavior of the WWW servers with "modified" UNIX kernel could be a possible solution. However, any approaches including kernel modifications requires the source code of the OS. Even if the source code is available, the modification of the kernel is hard because it requires skills and expertise.

The benchmark is another performance measurement method. For example, the *SPECweb96* [40] by The Standard Performance Evaluation Corporation and *WebStone* [91] are famous benchmark method for WWW servers. The benchmarks can provide various indices on the performance of the WWW servers. Especially, the benchmark is helpful to know the maximum performance of the WWW servers with various configurations. However, this method also has several drawbacks. The benchmark requires both the special benchmark software and their environment, and they are costly. Especially for the large scale WWW servers in operation, it is almost impossible to suspend their services only for benchmarks. If we can prepare them, the benchmark result can indicate the performance of the WWW servers in the environment specially prepared for the benchmarks. In other words, it is hard to apply the benchmark to the WWW servers that are in their actual operation.

As the method (4), the customer survey could be the other approach. In many WWW services, "customer feedback" type of questionnaires are prepared. This method is appropriate to know the level of customers' satisfaction, however, they cannot help the server improvement technically.

Consequently, the methods discussed above are not sufficient for the performance measurement of the "running" servers. It is obvious that a new method for this purpose should be developed.

The new method we propose here in this article is the performance measurement of WWW servers through the packet monitoring. The idea of this method is to know the behaviors of the WWW servers through the observation of the packets generated by both a WWW server and the WWW clients hooked up to the server. This method has several

advantages:

- Modification on OS kernels is not required.
- This method can be applied on any WWW servers even if the servers are in their actual operation.
- This method does not interfere between the WWW servers and the performance measurement tools.

However, our approach also has a drawbacks. Since our system cannot see inside of the WWW server, our system does not obtain any performance indices which are available in the OS inside. However, in the case we need to examine the WWW server internals, the method (1) or (2) can be used as a complement.

In the following sections, we discuss the details of our design and implementations of our system. Moreover, we show our case studies where we apply our system to several WWW servers.

4.3 Design of Our System

In this section, we describe the design of our system and show how we approach to the goal of our system.

4.3.1 The Performance Degradation Model

For the WWW server operators, it is quite important to know if the server performance is degrading. From our experiences on WWW server operations, we developed a simple model of the performance degradation of the WWW server. This model is a state transition model, shown in Figure 4.2.

In the state I, as an initial state, a WWW server can provide its service for all the requests without any overheads. The operators want the WWW server to stay at this state as long as possible.

If the performance of the server is degraded by some reasons, the state of the server moves to either the state II or III. In the state II, the server cannot accept all the request and some of the requests are rejected. If the server is in this state, we can observe:

- The server rejects some TCP [116] connection establishment requests because the "listen" queue in the socket layer is full.

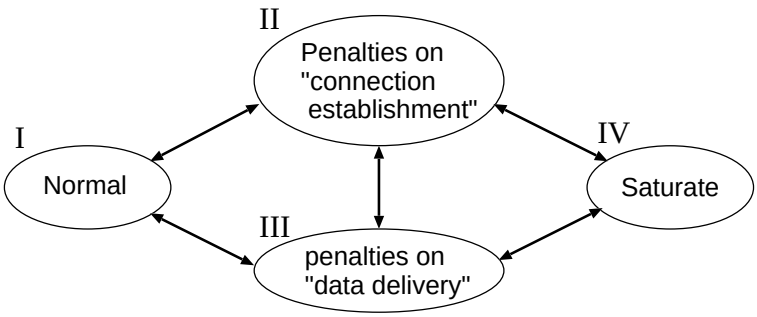

図 4.2: The performance degradation model

- The server accepts all the TCP connection establishment request, however, it rejects some HTTP requests. This can be caused by some specific server configurations.

In the state III, on the other hand, the data delivery from the server is sacrificed. The server can accept all the requests from the client, however, its data delivery becomes slower. This situation is frequently observed in the case we access busy WWW servers in the Internet. The HTTP connection is established but the we have to wait long time to obtain all the data in a WWW page.

The server may shift its state between the state II and III. However, if the situation is getting worse, the server's state is finally moved to the state IV. This state means that the server is saturated. In this state, the server rejects several accesses randomly, and the data is also slow to be delivered to the client.

Our system provides several performance indices in order to help the server operators to know these state transitions of WWW servers.

4.3.2 Performance Index

Through the packet monitoring, we can derive several performance indices. Based on the model discussed in the previous subsection, our system is designed to measure several performance indices listed below.

The Number of Concurrent Connections (Nc)

Normally, a WWW server can several HTTP connections in parallel. In the typical implementation of a WWW server on UNIX platforms, a single daemon process resides on the memory and is always waiting for HTTP request arrivals. *The Number of Concurrent Connections* means a set of HTTP connections that are handled concurrently. If the server

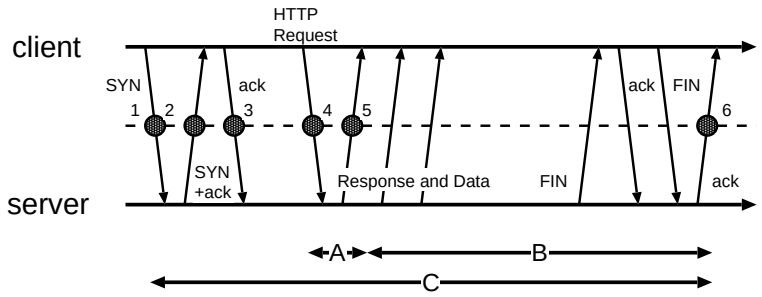

図 4.3: Performance Indices

is in the state II or IV, the number of concurrent connections is limited to the certain number or just decreased.

Connection Continuation Time (*Tc*)

Connection Continuation Time is defined as the period from TCP connection establishment to its shutdown. More precisely, the period from the time when the first SYN packet (the packet #1 in Figure 4.3) transmitted to a server by a client is observed to the time when the server's ACK packet (the packet #6 in Figure 4.3) corresponding to the last FIN packet generated by the client is monitored. The Connection Continuation Time in the state III or IV is much longer than one in the state I.

Note that this Connection Continuation Time is much longer than the connection time derived from either analysis on the WWW server log files. Almost all of the WWW server implementations, the time when the server calls *accept()* system call is recorded as the connection start time, and the connection end time is correspondent to the time when *close()* system call is processed. However, after the *close()* system call, the actual data delivery in the TCP layer is still going on, because the data is stored in the queue in the socket layer and the *close()* system call is immediately returned (see Figure 4.3).

Response Time (*Tr*)

The definition of *Response Time* is the period between an HTTP request packet from a client and its HTTP response packet from a server. However, this definition is slightly different from the generic definition of RTT (Round Trip Time). Since we discuss the behavior of WWW services through the packet monitoring, we use the period *A* in Figure 4.3. Therefore, the Response Time is normally shorter than the RTT, in general. This can be considered as a sign of the transition to the state III and IV.

Data Transfer Time (*Td*)

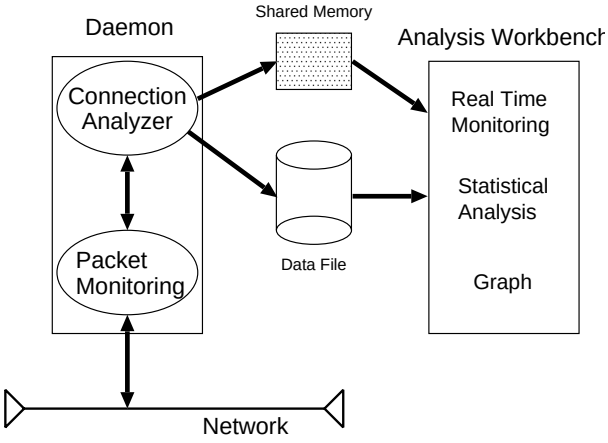

図 4.4: ENMA components

Data Transfer Time is defined as the period from the time when the first HTTP data packet transmitted by a server is observed to the time when the last data HTTP packet generated by a server is detected (The period B in Figure 4.3) . Note that the Data Transfer Time includes the period for exchanges of FIN packets and their ACK, because these packets sometimes include the actual WWW data. This can be also considered as a sign of the transition to the state III and IV.

Connection Setup Time (Ts)

We define *Connection Setup Time* as the period between when a server receives the first SYN packet (the packet #1 in Figure 4.3) and when the server sends a SYN+ACK packet back to the client (the packet #2 in Figure 4.3). In this period, the server allocates the memory for the socket, then put the SYN packet into the SYN-RCVD queue in the kernel [20]. Ts indicates the performance of the TCP/IP implementation and its platform.

4.3.3 Components

ENMA consists of two components: *ENMA Daemon* and *Performance Analysis Workbench* (see Figure 4.4)

ENMA Daemon

The ENMA Daemon is in charge of the "real time" packet monitoring. It is a single program which consists of two modules: a packet monitor and a connection analyzer.

The packet monitor captures all the packets in any HTTP connections observed on the network to which the ENMA system is attached. Since each captured packet is a data-link

frame, the packet capture strips a data-link header from the captured frame, and obtains an IP datagram in the frame. The obtained IP datagram is passed to the connection analyzer.

The connection analyzer is a core module of ENMA. The important function which the connection analyzer provides is to record the time when each packet is observed. Furthermore, the connection analyzer provides following functions:

- By the information in both IP and TCP headers in the packet, the connection analyzer distinguishes each TCP connection and tracks down its state transitions. This state transition analysis is used for calculation of performance indices listed in Section 4.3.3.
- The connection analyzer counts up the number of IP datagrams observed in each TCP connection. The total length of the payload in the IP datagram is also recorded. These measures are recorded separately for the upstream (from the client to the server) and the downstream (from the server to the client).
- The total length of the TCP payload is also recorded separately for the upstream and the down stream.

The connection analyzer writes these information to a single data file. This data file is used for the further statistical analysis by Performance Analysis Workbench. Note that the size of the data file is much smaller than files generated by either kernel-level monitoring or the command tcpdump, because a single data entry for a single TCP connection in the file is around 150 bytes. Therefore, we can use ENMA to monitor the WWW server for much longer.

The connection analyzer also provides the shared memory for other programs in Performance Analysis Workbench in order to enable them to make a realtime data visualization and analysis.

Performance Analysis Workbench

Performance Analysis Workbench is a set of programs which enable us to make various analysis on data that the ENMA Daemon provides. There are two kinds of programs in this workbench;

- The programs which make analyses on the data file provided by the ENMA Daemon. The statistical analysis tool is one of kind of programs. These programs read the data file and perform the data analysis in *batch* style.
- The programs which use the shared memory provided by the ENMA Daemon and make data analysis in realtime.

4.4 Implementation

We tried to implement our system as a portable system for several UNIX platforms. Currently, the ENMA Daemon was implemented in ANSI C using the GNU compiler (gcc) on FreeBSD 2.2.7. However, we can make it as a "platform-independent" program with using LBL's packet capture library. Programs in the Performance Analysis Workbench were implemented as shell scripts, AWK programs, and C programs. These program are also portable. In this section, we provide several technical notes on our implementation of the ENMA system.

4.4.1 Packet Monitor module

The packet monitor module in the current implementation uses the LBL's packet capture library (*libpcap* [76]) to capture the packets on the network. The library provides a common API for both the Berkeley Packet Filter (BPF [87]) on many BSD variants and the Network Monitoring Protocol (the packet snooper) on Sun's Solaris and SGI's IRIX.

4.4.2 Connection Analyzer

In the connection analyzer, we had to implement a module to monitor the TCP state transitions of each TCP connection. This module uses the sequence number, acknowledgment number, and flags in TCP headers for tracking down all the state transitions. The algorithm which we use in the connection analyzer is shown in Figure 4.5.

In this algorithm, the connection analyzer allocates a block of memories for each connection when its SYN packet is observed. This memory block is used as a memo for recording the state for each TCP connections. In the case the TCP connection is shut down normally, the memory block is released and the connection analyzer writes several performance measures listed in the Section 4.3 to the data file. However, there are several cases where the memory blocks are not released. These cases can happen in the following situations:

- The client tries to establish the TCP connection, however, the server rejects the request. In this case, since the client sends some SYN packets to the server, the connection analyzer allocates a memory block. This situation may cause by lack of system resources (e.g., the listen queue in the socket layer) in the server.

- The server has been clashed during the TCP connection is established. In this case, the client tries to keep the connection so that the client does not send any FIN or RST packets to the server. Therefore, the connection analyzer cannot release the memory block.

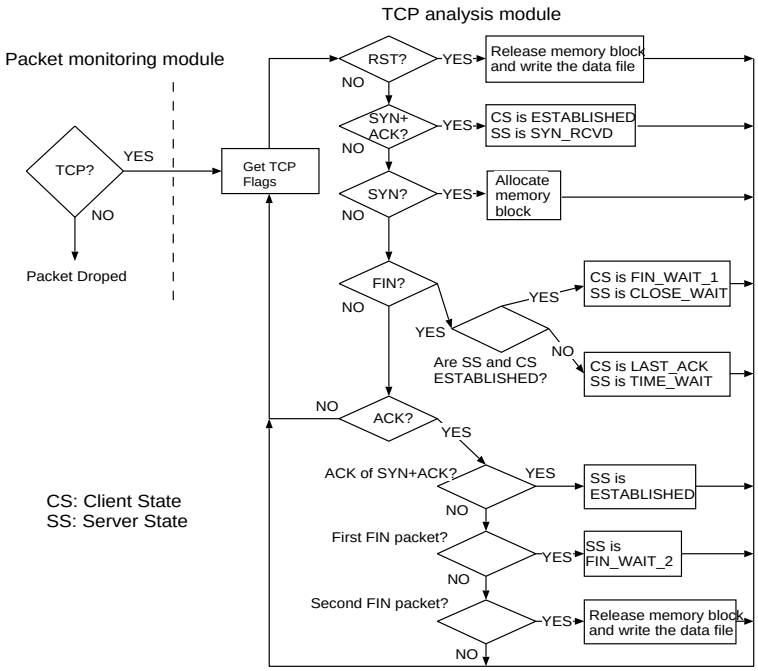

図 4.5: The connection analysis algorithm

- The packet monitor module misses captures of FIN and/or RST messages. In this case, a TCP connection was established but neither FIN nor RST packets are observed on the connection. Therefore, the connection analyzer cannot release the memory block.

In order to handle these cases listed above, the connection analyzer has a garbage collection mechanism for these "unreleased" memory blocks. In the current implementation, the connection analyzer tries to find any memory blocks that were not updated in the last 24 hours.

4.4.3 Performance Analysis Workbench

The Performance Analysis Workbench consists of two modules: the statistical data collecting module and the visualization module. All the modules are implemented in ANSI C.

The statistical data collecting module obtains the statistical data through shared memory. This module is used as an interface for other modules for "real-time" analysis. All the other modules have to obtain data through this statistical data collecting module. This module is quite simple and light weighted.

The visualization module is a X-window application program in order to view the data in various forms. The data is provided by the statistical data collection module through BSD's socket interface. Since the ENMA daemon consumes large amount of system resources so that running both the ENMA daemon and visualization program on the same machine may cause performance interferes. In the worst case, the ENMA daemon cannot dump all the packet from the network, therefore, the statistical analysis is less accurate. In order to avoid this situation, we implemented the Performance Analysis Workbench as separated modules.

4.5 Case Studies

In order to verify our implementation we applied our system to several "running" systems.

4.5.1 Case 1: The WWW Server

We measured the WWW server which provided various information and "live" video streams about the 80th National High-School Baseball Games of Japan in August, 1998. This event is very famous and popular in Japan so that the WWW server got over 32 million hits per day. We applied our system to this server for 16 days.

System Configuration

The target WWW server host was Sun Enterprise 450 server with dual CPU (300MHz Ultra SPARC processors) and 512MB of memory. The WWW server program was Apache 1.3.1 running on Solaris 2.6. This system was installed on the server segment (100BaseT). Our system was hooked up to the service segment and monitored the server.

Our system on which the ENMA daemon was running was an Intel platform (PentiumII 300MHz processor) with 64MB memory.

Evaluations

Figure 4.6 shows the frequency distribution of Connection Continuation Time (Tc). The analysis based on the log file generated by the WWW server reveals that Tc for the most connections is 1 millisecond, while 20 millisecond is from our ENMA system. As mentioned in Section 4.3.3, the result from the WWW log file is quite different from one from our ENMA system.

The log file generated by the WWW server is the activity logging of the WWW server as one of application programs on the system. In other words, the log file is an application level logging. The buffer size in the socket layer of Solaris 2.6 was configured as 8 Kbytes. Since almost all of the WWW objects that the server handled were under 8Kbytes as shown

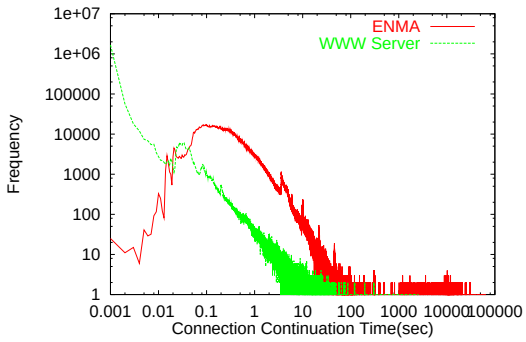

図 4.6: The frequency distribution of Connection Continuation Time

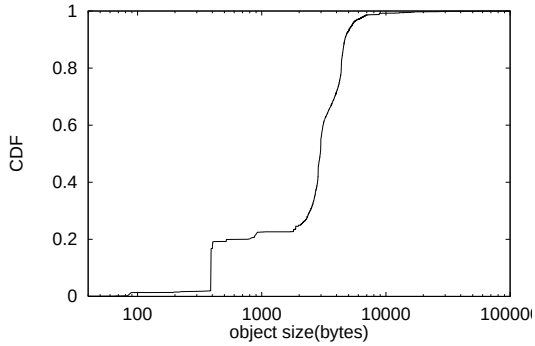

図 4.7: The cumulative distributed function of the WWW objects

in Figure 4.7, a single *write()* system call in the WWW server can put entire data of each WWW object into the socket buffer. Therefore, the server can process each HTTP request around 1 millisecond; the sequence of system calls [1] for request handling are processed and terminated immediately. *Tc* obtained from the log file by the WWW server means that the connection continuation time for each HTTP connection in the application layer.

On the other hand, *Tc* by ENMA includes the HTTP request processing plus its connection establishment and shutdown procedures in TCP layer. These requires at least 3 RTT. In other words, Tc derived by ENMA is the connection continuation time in the TCP layer. Therefore, the results reveal the differences.

The analysis on the number of concurrent connections (*Nc*) is shown in Figure 4.8 and 4.9. Since *Tc* by the WWW log file is different from one by ENMA, the analysis of *Nc* is also different; from the WWW server log file, 550 connection is in the peak, however, 13,000 connections were observed by ENMA.

As our results, therefore, the analysis through ENMA is more accurate than the result through the performance analysis on the WWW server's log files. Our result is more helpful to design and/or improve the network where the WWW server is located.

4.5.2 Case 2: Slow WWW servers

We conducted the other experiment for testing the performance measurement of the WWW server. As mentioned in Section 4.3, both the Response Time (*Tr*) and the Con-

[1] In the Apache server, each HTTP request is processed through the sequence of system calls : *accept()* for establishing the HTTP connection, *read()* for read HTTP request from the socket, *write()* for sending the WWW object, and *close()* for shutting down the connection.

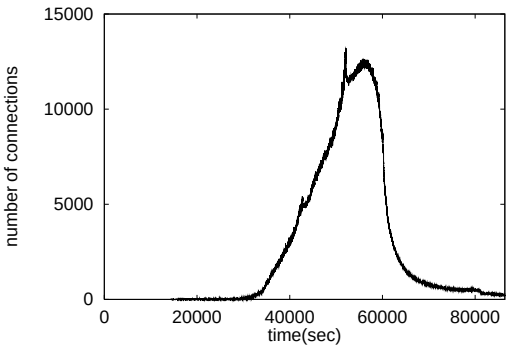

図 4.8: The number of concurrent connection by ENMA log

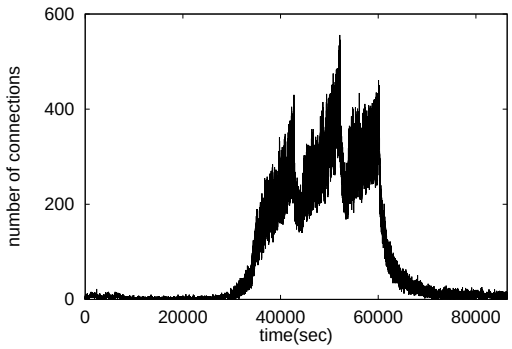

図 4.9: The number of concurrent connection by WWW server log

nection Setup Time (Ts) are expected as indices to reflect the performance of the WWW server.In this case study, we tried to confirm these value can be used as performance indexed of WWW server.

System Configuration

In this case study, we setup two WWW servers in our laboratory: the Apache [120] server running on Prntium II 200 MHz processor and one of the 80486DX2 66 MHz processor. The operating systems for these servers are FreeBSD 2.2.7. The benchmark software we developed is configured with the other system connected to the same network segment where the WWW server is located. The benchmark software is quite simple; the program tries to access several WWW objects on the WWW server at random. In this case, we measured Tc, Ts, and Tr for 10,000 accesses.

Results

Figure 4.10,4.11,4.12 show the results of our measurements. By these three graphs, we can easily read the differences on the performance between the WWW server on Pentium II and one on 80486DX2 66MHz. Our ENMA system can show the differences of WWW server's performance easily.

4.6 Discussions

We designed and implemented the ENMA system as discussed in the previous sections. However, our system is a kind of "alpha version" of the products. It is obvious that there are several limitations on our system as well as some extensions to improve our system.

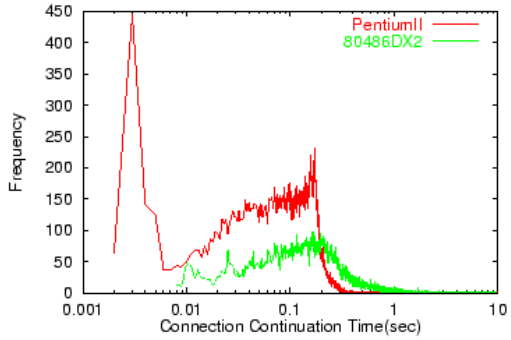

図 4.10: The frequency of the connection continuation time

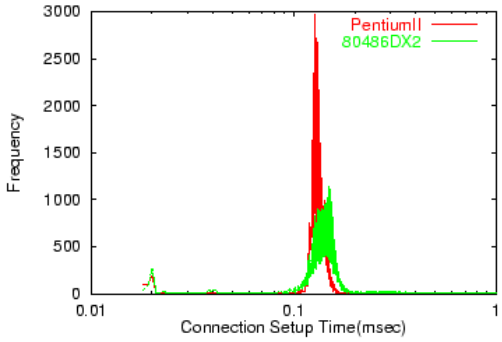

図 4.11: The frequency of the connection setup time

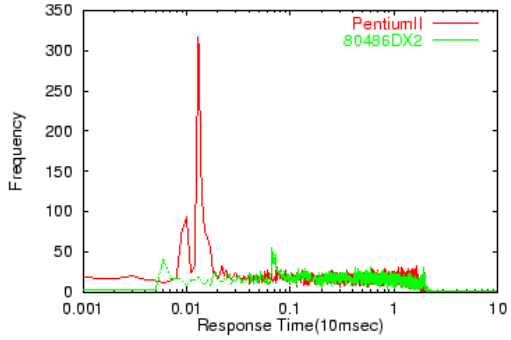

図 4.12: The frequency of the response time

4.6.1 Technical Issues

Dropping the packet through monitoring is a significant technical issue. Because of the design of ENMA system, dropping the packet at the monitoring may cause large influences on the performance analysis. Currently, the ENMA tries to grab all the packets as much as possible, however, there may be several packets dropped. There are solutions for decreasing the number of dropped packets:

- Using a faster computer with more memory as our ENMA system may improve the ENMA itself, however this solution cannot be used for any situations such as ENMA hooked up to Gigabit Ethernet or other higher bandwidth networks.
- Developing a new implementation of ENMA on lighter operating system such as DOS is possible solution. In general, the packet monitoring using UNIX operating

systems cause several problems in the case we apply them to the multi hundreds Mbps or more broadband networks. With this reason, the OC-3 packet monitoring tool called "OC3MON" [18] for example is implemented on DOS because of several considerations on OS overhead. Therefore, if we apply the ENMA to the Gigabit Ethernet or other high speed LAN, we have to re-implement ENMA on other lighter OS platforms.

The tracking down the sequence number in TCP header is the other technical issue. In the current implementation, ENMA does not handle the sequence number. Since changing the packet order may cause serious penalty on TCP performance, it is the better to track the sequence number in the TCP header.

The measuring the number of request rejected in TCP layer is the other technical issue. With the heavy loaded WWW server, some SYN packets are once received by the WWW server system but rejected at TCP layer. These phenomena is frequently observed at heavy loaded servers. It is better to monitor these phenomena by ENMA to allow the system managers to know if the system is saturated.

4.7 Concluding Remarks

In this paper, we describe several reasons why the new method for measuring the performance of the WWW server is required. Our proposed method is based on the packet monitoring to reveal all the behavior of the WWW server and derive several performance indices through the monitoring. The method has been implemented as our ENMA system. The ENMA can measure performance indices such as Connection Setup Time, Connection Continuation Time, The Number of Concurrent Connections, etc. As mentioned in Section 4.5, we applied ENMA system to several WWW servers and confirmed the efectiveness of its implementation.