Regular Paper

A Framework for Gigabit-rate Packet Header Collection to Realize Cost-effective Internet Monitoring System

Teruyuki Hasegawa,† Tomohiko Ogishi† and Toru Hasegawa†

In this paper, we present a framework for realizing low-cost Internet monitoring systems that can handle gigabit-rate traffic without any special hardware. As a result of the spread of Gigabit Ethernet technologies, there is a need for inexpensive solutions that can realize Internet monitoring systems with gigabit-rate capability. However, it is quite difficult for existing software-based systems to collect all the packet headers at a gigabit rate, even if high-end hardware components are used. We therefore propose a novel framework for gigabit-rate packet header collection that can be applied to most network interface cards. This framework also provides a capability for forwarding the collected header information to other hosts via a network, which makes it easy to integrate existing applications and to introduce load-balancing mechanisms. The results of our performance evaluation show that our first implementation is capable of collecting 100% of header information, even if a Gigabit Ethernet link is fully utilized by 384-byte packets, while existing software-based systems can collect less than 50%, even though the CPU load is more than twice as large.

1. Introduction

Recently, the traffic volume in the Internet has been growing rapidly because of the continuous increase in the number of users and the development of various so-called *broadband* access media. In order to accommodate the large volume of traffic, many Internet service providers or enterprise networks have been using *Gigabit Ethernet* (*GbE*) [1] to build local-area backbone networks. GbE is also beginning to be utilized as an access link to wide-area IP backbone networks.

As a result of the spread of GbE technologies, there is a need for reasonable methods of constructing Internet monitoring systems with gigabit-rate capability, so that network operators or users can obtain various information about networks such as network usage and workload trends, or obtain the observed packets themselves. From the viewpoint of cost and flexibility, it is very desirable to realize Such monitoring systems by using commodity hardware such as combinations of personal computers (PCs) and network interface cards (NICs) on which some existing software is running. However, it is quite difficult for these software-based systems to collect all the packet headers at a gigabit rate, even if a high-end PC system is adopted.

In order to keep pace with gigabit-rate traffic, we propose a novel framework for packet header collection to realize cost-effective Internet monitoring systems. This framework is a completely software-based solution and can be applied to most modern NIC designs with direct memory access (DMA) capability. The main features of our framework are as follows:

- In order to attain gigabit-rate performance, a special software module is introduced as a part of the device driver of a GbE NIC. This module extracts only header information from received packets into a contiguous kernel memory space, utilizing DMA transfer from the NIC.
- This module can encapsulate a group of collected header information into a UDP message and forward it to remote PCs via a network, as well as to the local PC. Because of these capabilities, it is easy to introduce load-balancing mechanisms. By using this load-balancing mechanism, an Internet monitoring system with gigabit-rate performance is achieved with a few PCs.

The rest of this paper is organized as follows. Section 2 summarizes some related work on packet collection. Section 3 enumerates some design principles of our framework and presents an overview and details of the design. Section 4 describes a sample implementation of the header collection system using the framework. Section 5 gives some results of its performance evaluation. Sections 6 and 7 consist of a discussion and conclusions, respectively.

† KDDI R&D Laboratories, Inc.

2. Related Work

This section surveys existing frameworks for packet or packet header collection using commodity hardware. We categorize them into the following two approaches:

2.1 Approach with OS Functions

In order to realize a monitoring application at the user-level, many operating systems (OSs) contain some function applicable to user-level packet collection. The application can interact with NIC hardware and receive *raw packets* transferred over a network directly from the NIC.

The *BSD Packet Filter* (*BPF*) [2] is the best-known framework for providing such a function. It is implemented as a kernel component of BSD-based Unix OSs (e.g., FreeBSD and NetBSD). **Figure 1** shows an overview of BPF. In this framework, each link-layer device driver (e.g., a driver for Ethernet NICs) arranges the *network tap* to call the BPF tap function. BPF associates a filter and a pair of buffers in the kernel with each application. The conditions of a filter are given by each application via an *ioctl* system call. One of these buffers, called the *Store Buffer*, is used to receive packets from the NIC. The other, called the *Hold Buffer*, is used to copy the packets to the application. BPF swaps these buffers when the Store Buffer is full and the Hold Buffer is empty.

On the arrival of a new packet from a NIC, a device driver first calls the tap function before sending the packet up to the system protocol stacks if there is any application listening on this NIC via BPF. The tap feeds the packet to the filter. Next, the tap copies a part of the packet (or in some cases the whole packet) accepted by the filter to the Store Buffer. The device driver then regains control. Since BPF blocks the *read* system call from the application while the Hold Buffer is empty or until some buffering timeout has occurred, the application can read several packets at a time. This kernel buffering mechanism reduces the number of expensive user-kernel interactions and extra memory copies.

In SVR4-based Unix OSs (e.g., Solaris), such a function (i.e., raw packet operation) is covered by the *Data Link Provider Interface* (*DLPI*) [3] supported in link-layer device drivers based on STREAMS [4]. The Linux OS provides the *packet socket* [5] mechanism for this purpose. For the Win32 family of OSs (e.g., Windows 95, 98, NT, and 2000), a similar approach can be used according to the *Network Driver Interface Specification* (*NDIS*) [6]. These frameworks have the following features:

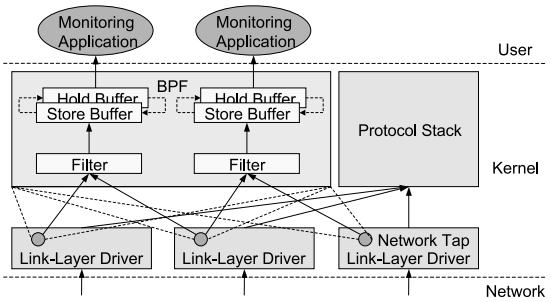

Fig. 1 Overview of BPF.

- Unlike BPF, these three mechanisms do not provide kernel buffering by themselves. This may cause per-packet-based user-kernel interactions and extra memory copies.
- In DLPI and NDIS cases, it is possible to introduce some additional kernel modules sitting on top of a device driver to support kernel buffering. For example, the *bufmod* is provided in Solaris, and the *Packet Capture Driver* has been developed as a kernel-side component of the *Windump* [7] system for the Win32 family. In these approaches, it is necessary for device drivers to copy (or duplicate) the whole part of every received packet in order to pass it up to these modules in addition to system protocol stacks. This is because these modules should be scheduled independently of the device driver's context. Consequently, some overheads may still remain.

To provide a uniform method for user-level packet collection concealing OS-dependent interfaces as described above, *libpcap* [8] has been developed for a wide variety of UNIX OSs, and is utilized in many monitoring applications such as *TCPdump* [9].

2.2 Approach with Hardware Customization

When 155 Mbps OC3/ATM-based networks appeared, the above approaches did not scale for such a high rate at first, because the CPU performance was insufficient. In order to collect all packet headers without heavy CPU loads, *OC3MON* [10] and its OC12 version *OC12MON* were developed and ported to Unix (*Coral Reef* [11]). **Figure 2** shows overviews of them.

OC3MON adopts the following approaches to

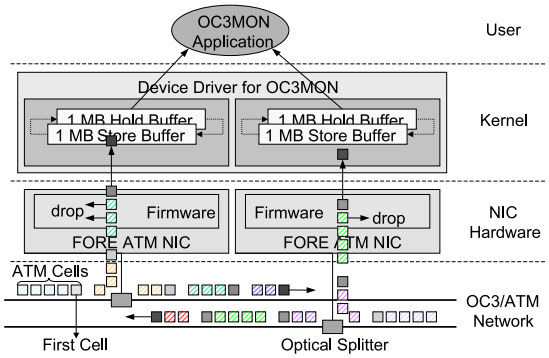

Fig. 2 Overview of OC3MON.

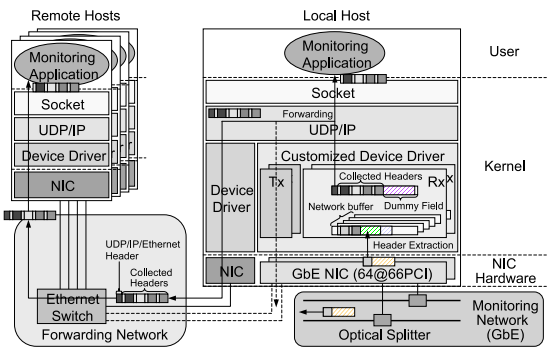

Fig. 3 Overview of proposed framework.

keep pace with the OC3 line rate.

- It uses an ATM NIC with an on-board CPU for each direction. Unlike the above approaches, this ATM NIC is used only for packet capturing.
- A special firmware running on NIC's CPU is introduced. It is customized to select the first cell of an AAL5 PDU received from an OC3/ATM line and forward it into a contiguous kernel memory space in the host with a timestamp. A cell accumulated in the host memory is expected to contain a TCP/IP header. This reduces the volume of DMA transfer between the NIC and the host.
- After a fixed number of cells have been transferred, NIC's CPU interrupts the host CPU to inform the arrival of these cells. This avoids the occurrence of frequent interruptions, which tend to be a heavy load on the host CPU.

3. Framework Design

In this section, we explain our framework design for packet header collection aiming at higher performance than the approach with OS functions such as BPF without any hardware-level customization.

3.1 Principles

We adopted the following design principles for the framework:

(1) The framework should be designed so that it can be applied to various GbE NICs including future products. In other words, the NIC customization adopted in OC3MON's approach is not used, because this does not provide any scope for choosing the NIC. This will make it possible in the future to use more powerful NICs than those available today.

(2) According to some research on analysis of Internet backbone traffic [12),13)], the mean IP packet size is in the range between 384 and 512 bytes. Header collection should be realized at the full line rate, even if the mean packet size is less than 512 bytes.

(3) The host CPU is required to handle about 230,000 packets per second in each direction if a GbE link is fully occupied by 512-byte IP packets. At such a packet rate, some optimized kernel-buffering mechanism is indispensable in order to save the CPU load.

(4) The header collection may be so heavy that there is no room for executing other monitoring functions in the same host. The header collection thus needs to be executed independently of the other monitoring functions. This means that it is necessary for some function to forward collected headers to other hosts. Obviously this *header forwarding* should be provided without significant processing overheads.

3.2 Overview

Figure 3 shows an overview of our framework. In accordance with the above principles, header collection is realized by a combination of a 64-bit, 66-MHz PCI GbE NIC with DMA (i.e., PCI Bus Master) capability and a customized device driver. Although this NIC is used only for the header collection on the receiving side, it is still possible to transmit a packet normally.

The driver customization is applied to receive side routines so that the first N-byte field of every received packet can be extracted to the data region of an OS-specific network buffer (e.g., *mbuf* [14)] in BSD, *sk_buff* [15)] in Linux) contigu-

ously. Here, the value of N is configured by a user in 8-byte multiples according to the purpose of the monitoring application. For example, $N = 56$ bytes is recommended for 40-byte TCP/IP header collection if a 14-byte Ethernet header is also transferred from the NIC.

This driver also provides capabilities of encapsulating a group of collected header information into a UDP message and forwarding it not only to the local host but also to remote hosts via a network (*forwarding network*) separated from the network doing the monitoring (*monitoring network*). The monitoring applications located on either local or remote hosts can receive this aggregated header information through the UDP/IP stack.

3.3 Details

3.3.1 NIC Requirements

With regard to the header collection, the requirements for GbE NICs have been narrowed down to just the following two items so that this framework can be applied to various NIC implementations.

(1) Since our framework needs some modification of the NIC's device driver, it is mandatory that the driver source code should be available.

(2) The NIC must provide DMA capability whose target host memory addresses can be controlled by the device driver on a per-packet basis.

Figure 4 shows an example of host-NIC interface definition on the receiving side, which satisfies the second requirement. As far as we know, most GbE NICs use similar interfaces. In Fig. 4, the device driver arranges several pairs composed of a packet buffer (*Buffer* for short) and its descriptor (*Buffer Descriptor, BD* for short). A Buffer is a physically contiguous memory space in which a whole packet (or occasionally a part of a packet) is accommodated. Ordinarily, the data region of an OS-specific network buffer is used as a Buffer. A BD is associated with a Buffer to maintain its parameters such as the *Baddr* and the *Blen* for DMA transfer. These parameters indicate the physical address and the length of the associated Buffer, respectively. A received packet is transferred from the NIC to the host according to the following sequence:

(1) The device driver arranges several Buffer-BD pairs in advance. BDs are registered with the free BD queue, which is accessible from the NIC via a PCI bus.

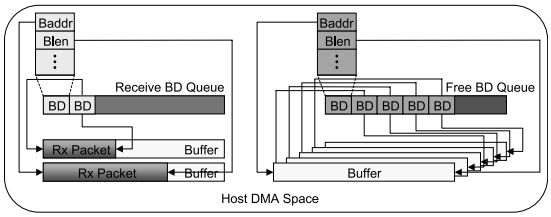

Fig. 4 Host-NIC interface example (receiving side).

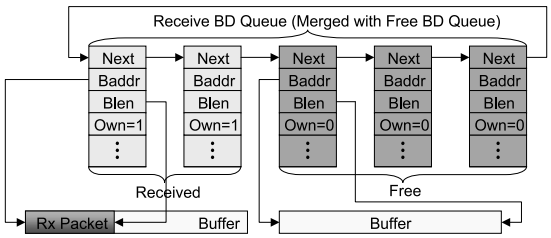

Fig. 5 Chained BD queue structure.

The Baddr and the Blen fields in BDs are set up to inform the NIC of the starting address and the length limitation of DMA transfer.

(2) The NIC reads a BD from the free BD queue to determine where and how long a received packet can be transferred. On arrival of a packet, the NIC consumes a free BD and feeds the packet to a Buffer associated with the BD. The Blen field in the BD is updated according to the packet length.

(3) After this DMA transfer is completed, the NIC moves the BD from the free BD queue to the receive BD queue. Then the NIC generates a receive interrupt directed to the host so that the Buffer-BD pair will be processed in the device driver.

It should be noted that some NIC implementations adopt the *chained BD queue structure*, in which several BDs constitute a single linked list, as shown in **Fig. 5**. Although the free BD queue is merged into the receive BD queue in this structure, the host and the NIC can distinguish both queues using the *Own* flag arranged in each BD. Thus we consider this structure to be identical in nature with the one illustrated in Fig. 4.

On the other hand, there is no requirement for NICs with regard to the header forwarding. Even the use of GbE NICs is not necessarily assumed because the header forwarding rate may be lower than the gigabit rate.

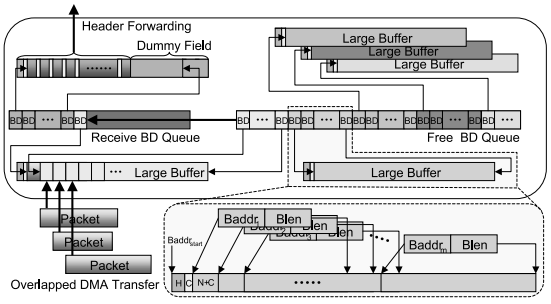

Fig. 6 Header extraction mechanism.

3.3.2 Header Extraction

Figure 6 shows how packet headers can be collected without any extra memory-to-memory copies on the host. Our framework adopts the following methods to extract multiple headers contiguously to a Buffer:

(1) The device driver arranges several large Buffers. Each Buffer is associated with multiple BDs in advance so that multiple packets can be DMA-transferred into a common Buffer. We assume that the number of BDs per Buffer is m in the following description.

(2) The Baddr field of each BD is configured with some gap. That is, let i ($1 \leq i \leq m$), $Baddr_i$, and $Baddr_{start}$ be the order of BD, the Baddr of the ith BD, and the start address of the Buffer, respectively. The value of $Baddr_i$ is determined as follows:

$$Baddr_1 = Baddr_{start} + H + C$$
$$Baddr_i = Baddr_{i-1} + N + C \quad (i \geq 2),$$

where N and C indicate the length of header information (see Section 3.2) and that of the control field (if any) located just before the header information, respectively. H is determined on the basis of the length of the UDP/IP/Ethernet header adding in the header forwarding (see Section 3.3.3).

(3) On the other hand, every Blen field is filled with a fixed value, which is sufficiently large for the associated BD to accommodate a whole Ethernet frame (i.e., an IP packet with Ethernet header). For example, a Blen field must be equal to or larger than 1514 bytes excluding the 4-byte CRC field, if standard Ethernet frames are used. The real Buffer size $Bsize_{real}$ must satisfy the following equation:

$$Bsize_{real} \geq m \times (N + C) + H + Blen.$$

(4) Along the sequence described in Section 3.3.1 (2) and (3), a new packet is DMA transferred from the NIC overlapping with the previous one on a common Buffer, except for the first N plus C bytes of the previous one. The latter C bytes are reused for the control field of the new one. As a result, the first N bytes of every packet (i.e., packet header) are automatically gathered in a contiguous memory space without CPU loads.

(5) When a receive interrupt occurs, the device driver reads a BD from the receive BD queue and coordinates the corresponding control field. The contents of the control field are specified according to the target monitoring applications (see Section 4). It should be noted that the Ethernet header field is allowed to be reused as the control field if the Ethernet header is not a monitoring target.

(6) After completing the above task for the mth packet, the device driver feeds a Buffer filled with collected headers up to the IP layer according to the sequence described in Section 3.3.3.

3.3.3 Header Forwarding

Before starting the header collection, the user needs to decide the following parameters related to the header forwarding.

- UDP port numbers (source and destination)
- IP addresses (source and destinations)
- IP packet size

Here, the UDP port number pair is common to all the monitoring applications. The source IP address is a dummy one which must be preassigned for the header forwarding. The destination IP address must be set for every destination host regardless of whether the destination is local or remote. The IP packet size should be smaller than the MTU (maximum transfer unit) size of the forwarding network. However, in order to perform the header forwarding efficiently, it is necessary to use a large IP packet size (e.g., 4 KB). Such a requirement is satisfied if forwarding packets are directed only to the monitoring application in the local host, or if jumbo frames [16] are supported on the forwarding network. In both cases, the user must set an adequate MTU size for all the NICs prepared for the header collection and the header forwarding.

In addition, it is recommended that the user

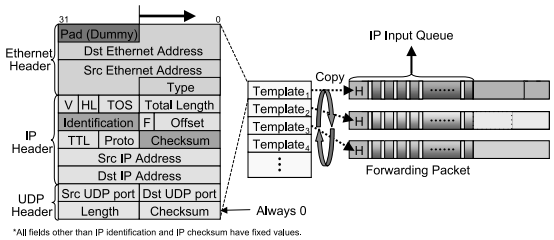

Fig. 7 Header forwarding procedure.

creates static ARP (address resolution protocol) [17] entries for remote hosts in advance so that the local host can output an IP packet without an address resolution process which associates the IP address with an Ethernet address according to ARP. It should be noted that this static ARP entry creation is mandatory if the header forwarding is also executed on the GbE NIC prepared for the header collection. This is because the receiving side of this NIC does not attach the forwarding network and therefore cannot receive ARP reply messages from this network.

Figure 7 illustrates the header forwarding procedure. The header forwarding is realized as follows:

(1) The device driver allocates UDP/IP/Ethernet header templates for the header forwarding according to the number of destination hosts. The contents of these templates are configured in advance on the basis of the above parameters, because the values of most header fields do not need to be changed.

(2) In a template, the IP checksum value is set to a pre-calculated one to realize IP checksum generation with minimum overheads. On the other hand, the UDP checksum value is set to 0 in order to eliminate expensive UDP checksum verification performed by the UDP in the destination hosts, as well as its generation performed by this forwarding procedure.

(3) When a Buffer has been filled with collected headers as described in Section 3.3.2, one of the above templates is selected in turn, then the selected template is adjusted and copied into the first H bytes of the Buffer to be recognized as a forwarding packet.

(4) On the adjustment, the IP ID (Identification) value in the template is incremented by 1 because the forwarding packets should have different IP IDs from each other. As a result of this adjustment, the IP checksum value must be decremented by 1.

(5) Finally, the forwarding packet is registered with an IP input queue as if a normal UDP/IP packet had just arrived. The packet is forwarded to an appropriate destination by the IP.

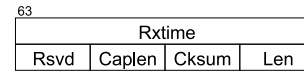

Fig. 8 Control field format.

4. Implementation

We have implemented a header collection system based on the above framework as a sample system. The system was developed for PCs with Linux 2.4 OS, because this provides the most various GbE NIC options as regards the condition enumerated in Section 3.3.1 (1). There are two main components of the system: the *header collecting driver* and the *file recording application*.

The header collecting driver was developed on the basis of the OS-bundled device driver for the SysKonnect SK-NET GE family of GbE NICs [18] with some customization according to the description in Section 3. For the control field format added to each collected header (see Section 3.3.2 (5)), we specified the following 16-byte fields, as shown in **Fig. 8**:

- ***rxtime***: a 64-bit value for recording the arrival time of the packet in nanoseconds. The driver generates this value from either the system time or the hardware timestamp provided by the NIC. This hardware timestamp is a 32-bit value based on the 32,768-kHz on-board clock [19].
- ***rsvd***: a 16-bit reserved field for future use.
- ***caplen***: a 16-bit value indicating the extracted data (i.e., header) length of the packet.
- ***cksum***: a 16-bit value proving the 16-bit 1's complement checksum of the whole IP packet calculated by the NIC. This field assists monitoring applications to verify the TCP/UDP checksum [20] of the packet. The applications need only to subtract some IP field values in the collected header from this cksum value. If the calculated value is equal to 0xFFFF, the corresponding TCP/UDP checksum is correct.

- ***len***: a 16-bit value of the total packet length including non-extracted data.

The driver also supports two additional types of format in order to work together with existing monitoring applications. One is used for libpcap-based applications such as TCPdump. The other is used for our proprietary *Internet Performance Monitor* [21] application.

On the other hand, the file recording program is a user-level application that receives a UDP message filled with collected headers from the UDP/IP stack through the socket interface [22] and stores it in file systems. This application can be executed on either local or remote hosts.

5. Performance Evaluation

In order to evaluate the proposed framework, we measured the performance of our header collection system, using the configuration shown in **Fig. 9**. We arranged two PCs (PC1 and PC2 in Fig. 9) with 64-bit, 66-MHz PCI slots as a platform of our system. **Table 1** shows the hardware specifications of these PCs. The header collecting driver is introduced in PC1, which receives packets on the monitoring network via an optical splitter. PC2 is connected to PC1 through the forwarding network. The file recording application is working on either PC1 or PC2 in order to evaluate the effect of the header forwarding to remote hosts. The traffic on the monitoring network is emulated by a commercial traffic tester (ANTARAnet FLAME THROWER [23]), which can generate test traffic at the full line rate of GbE in any Ethernet frame size.

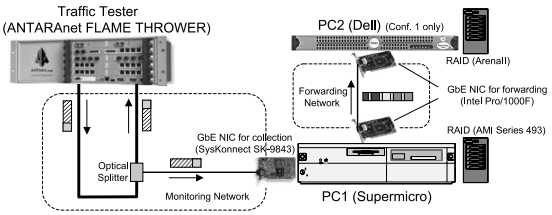

Fig. 9 Network configuration.

In each test, the traffic tester transmits about 10,000,000 Ethernet frames in one direction at the full line rate of GbE. The Ethernet frame size is fixed during a test and set to 64, 128, 256, 384, 512, 768, 1,024 or 1,518 bytes. The first 40-byte fields of IP packets (i.e., excluding Ethernet headers) are collected using various system configurations, as shown in **Table 2**, where TCPdump is adopted as an existing system for comparison with our proposed system. TCPdump is executed with the "-w" option by which the collected headers are simply stored in file systems, without being analyzed or displayed. In Confs. 4 and 6, *null device* (*/dev/null*) is used in order to avoid some overheads of accessing real file systems for comparison with Conf. 1, where no file access occurs in PC1. In addition, we increase the buffer size of BPF or the socket to 4 MB using the *sysctl* command, so as not to drop the collected headers. The MTU size assigned to every NIC is 4,124 bytes where 73 headers can be aggregated in a forwarding packet.

As the performance indexes, we measured the *collection ratio* and the *CPU load*. The former indicates the ratio of the number of collected

Table 1 Hardware specification.

	PC1	PC2
Vendor	Supermicro	Dell
Chipset	ServerWorks HE-SL	
CPU	Intel Pentium III 1 GHz × 2	Intel Pentium III 866 MHz × 1
Memory	512 MB	128 MB
RAID system	AMI Series 493 with IBM DDYS-T36950 × 4	ArenaII with IBM DTLA-307075 ×4
NIC (collection)	SysKonnect SK-9843	N/A
NIC (forwarding)	Intel Pro/1000F	

Table 2 System configuration.

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6
OS	Linux 2.4.2				FreeBSD 4.2	
Collection mechanism	Proposed		Packet socket		BPF	
Collection platform	PC1					
Application (AP)	Proposed		TCPdump with libpcap			
File system (FS)	RAID			/dev/null	RAID	/dev/null
AP and FS location	PC2	PC1				

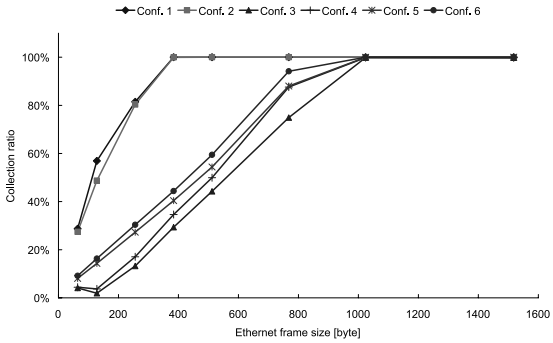

Fig. 10 Collection ratio.

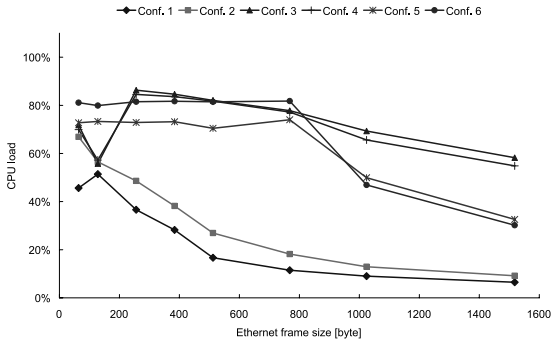

Fig. 11 CPU load at PC1.

packets to the number of transmitted packets on the monitoring network. The latter is the average value of CPU loads reported by the *top* command every 1 second. **Figures 10**, **11**, **12** show the results of the collection ratio, and the CPU load observed on PC1 and PC2, respectively.

6. Discussion

(1) As shown in Fig. 10, the proposed header collection system achieved higher performance than existing TCPdump-based systems. In Conf. 1, our system can collect 100% of header information at the full line rate of GbE in the case of 384-byte and larger Ethernet frame sizes. In Conf. 2, our system attains a 100% collection ratio when the frame size is equal to or larger than 512 bytes. In contrast, existing systems using Linux (Confs. 3 and 4) drop some packets even in the case of a 1,518-byte frame size. In FreeBSD cases (Confs. 5 and 6), where BPF's kernel buffering is applied, the performance is somewhat better than in the Linux cases, but packet loss still remains when 784-byte (in Conf. 6) or 1,518-byte (in Conf. 5) frames are collected. These results confirm that our framework is very effective in improving the performance of header collection and provides sufficient capabilities for monitoring Internet backbones where the mean IP packet size is in the approximate range between 384 and 512 bytes (see Section 3.1 (2)).

(2) Figure 11 shows that our framework can also reduce the CPU load dramatically as compared with existing frameworks. For example, compared with Conf. 1 and Conf. 6 in the case of a 384-byte frame, our system collects 100% of headers with a 28.2% CPU load in Conf. 1, while the existing system collects 44.4% of headers with an 81.7% CPU load in Conf. 6. In the case of 1,024 bytes, where both configurations can collect 100% of headers, 9.0% and 46.9% CPU loads are observed, respectively. This improvement is very significant, especially in the case where a target monitoring application adopts some sophisticated procedures. For example, the Internet Performance Monitor [21] performs real-time TCP behavior analysis for collected headers. This real-time analysis is considered to require a lot of CPU power at gigabit rates.

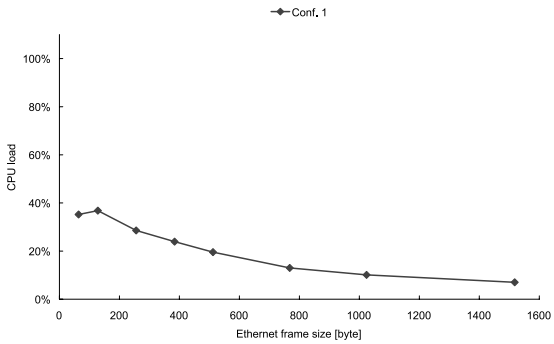

Fig. 12 CPU load at PC2 of Conf. 1.

(3) According to Fig. 10, the header forwarding to a remote host (PC2) is effective in improving the collection ratio at PC1. For example, in the case of a 128-byte frame size, the collection ratio observed on PC1 (local host) is 56.9% in Conf. 1. In Conf. 2, the value is 48.7%.

On the other hand, Figs. 11 and 12 show that the CPU load at PC2 (remote host) in Conf. 1 is always smaller than that at

PC1 in Conf. 2 in spite of the lower CPU performance in PC2. This implies that the CPU load for the header collection can be isolated from a monitoring application, and that we can assign more CPU power to the monitoring application by use of the header forwarding to remote hosts.

If further reduction of header collection overheads from a monitoring application is required, it is possible to introduce some mechanisms such as M-VIA [24] where the forwarding packets can bypass kernel-level operations such as UDP processing and can be delivered directly to user-level applications without memory copies between user-space and kernel-space. However, this is an application-side matter, and is beyond the scope of our framework.

(4) With regard to the efficiency of PCI bus and CPU utilization, our approach is less optimized than OC3MON [10], because a whole packet is transferred via PCI bus and some per-packet-based operations are still remain in our approach, while OC3MON avoids them (see Section 2.2). However, we did not choose OC3MON's approach, and developed a fully software-based approach for GbE monitoring because of the following reasons. First, as far as we know, there is no commodity GbE NIC that provides on-board programming facilities and sufficient performance to receive small packets simultaneously. Second, it is expected that 64-bit, 66-MHz PCI provides sufficient bandwidth for receiving whole packets at a gigabit rate and forwarding their headers. Moreover, advanced I/O technologies such as *PCI-X* [25] or *InfiniBand* [26] are ready or will soon appear: for example, some commodity GbE NICs support PCI-X. We therefore believe that we should not limit the scope for NIC choice.

(5) At the header forwarding, our framework design omits UDP checksum generation and verification in view of its performance penalty. Consequently, a forwarding packet filled with collected headers may be corrupted. However, we believe that there is very little possibility of the forwarding packet being corrupted in our system for the following reasons. First, if a monitoring application is located on the local host, the forwarding packet is exchanged within the same host. Thus, no verification mechanism is required. Second, in the case that monitoring applications are executed on the remote hosts, we assume that the remote hosts are attached to the same Ethernet segment. Since Ethernet provides a 32-bit CRC mechanism, the corruption of the forwarding packets is detectable.

On the other hand, if a user requests UDP checksum generation and verification, we think that such calculations should be performed in the NIC on a hardware basis. This function is supported by most GbE NICs and several Fast Ethernet NICs.

7. Conclusion

This paper has described a framework for composing low-cost Internet monitoring systems that can handle gigabit-rate traffic without any special hardware. In order to support gigabit-rate traffic, we propose a novel framework for packet header collection that can be applied to most commodity GbE NICs. According to this framework, multiple packet headers are automatically collected in a contiguous kernel memory space utilizing an NIC's DMA transfer mechanism without any extra memory copy on a host. The framework also provides a capability for forwarding the collected header information to other hosts via a network, as well as the local one, which makes it easy to introduce load-balancing mechanisms. By means of this load-balancing mechanism, an Internet monitoring system with gigabit-rate performance is achieved using a few PCs. The results of our performance evaluation show that our first implementation can collect 100% of header information even if a GbE link is fully utilized by 384-byte packets, while existing *TCPdump* systems can collect less than 50%, although their CPU loads are more than twice as heavy.

Acknowledgments The authors wish to thank Mr. T. Asami, President & CEO of KDDI R&D Laboratories Inc., for his continuous encouragement in this study.

References

1) IEEE Computer Society: IEEE Std 802.3, 2000 Edition (Oct. 2000). http://standards.ieee.org/getieee802/.
2) McCanne, S. and Jacobson, V.: The BSD Packet Filter: A New Architecture for User-level Packet Capture, *Proc. 1993 Winter USENIX Technical Conference* (Jan. 1993).
3) UNIX International: *Data Link Provider Interface (DLPI) Specification: Revision 2.0.0*, OSI Work Group (Aug. 1991).
4) UNIX Software Operations: *UNIX System V Release 4 Programmer's Guide: STREAMS*, Prentice Hall (1990).
5) PACKET(4), Linux Programmer's Manual (Oct. 1998).
6) Microsoft Corporation: NDIS Packet Driver 3.0 (1996).
7) Loris, D., Viano, P. and Risso, F.: *WinDump: TcpDump for Windows*, Politecnicio di Torino, Italy (2000). http://netgroup-serv.polito.it/windump/.
8) McCanne, S., Leres, C. and Jacobson, V.: *LIBPCAP*, Network Research Group, Lawrence Berkeley National Laboratory. ftp://ftp.ee.lbl.gov/libpcap.tar.Z.
9) Jacobson, V., Leres, C. and McCanne, S.: *TCPdump 3.4*, Lawrence Berkeley National Laboratory, Berkeley, CA (June 1998). http://www-nrg.ee.lbl.gov/.
10) Apisdorf, J., Claffy, K., Thompson, K. and Wilder, R.: OC3MON: Flexible, Affordable, High-Performance Statistics Collection, *Proc. INET'97* (June 1997). http://www.nlanr.net/NA/Oc3mon/.
11) Keys, K., Moore, D., Koga, R., Lagache, E., Tesch, M. and Claffy, K.: The Architecture of CoralReef: An Internet Traffic Monitoring Software Suite, *Proc. PAM2001* (Apr. 2001). http://www.caida.org/outreach/papers/pam2001/coralreef.xml.
12) Claffy, K.: The Nature of The Beast: Recent Traffic Measurements from an Internet Backbone, *Proc. INET'98* (July 1998). http://www.caida.org/outreach/papers/Inet98/.
13) McCreary, S. and Claffy, K.: Trends in Wide Area IP Traffic Patterns, *A View from the Ames Internet Exchange* (May 2000). http://www.caida.org/outreach/papers/AIX0005/.
14) McKusick, M.K., Bostic, K., Karels, M.J. and Quarterman, J.S.: *The Design and Implementation of the 4.4BSD Operating System*, Addison-Wesley, Reading, MA (1996).
15) Cox, A.: Network Buffers and Memory Management, *Linux Journal*, Issue No.30 (1996). http://www2.linuxjournal.com/lj-issues/issue30/1312.html.
16) Alteon Web Systems: Extended Frame Sizes for Next Generation Ethernets, http://www.alteonwebsystems.com/products/whitepapers/jumboframes/.
17) Plummer, D.: An Ethernet Address Resolution Protocol, STD 37, RFC 826 (Nov. 1982).
18) SysKonnect: SK-NET GENESIS ⟨x⟩ Technical Manual (Nov. 1998). http://www.syskonnect.com/.
19) XaQti Corporation: XaQti XQ11800FP 1000 Mbps Gigabit Ethernet Controller Data sheet Rev. 8 (Sep. 1998).
20) Raden, R., Borman, D. and Partridge, C.: Computing the Internet Checksum, RFC 1071, ISI, Cray Research, BBN Laboratories (Sep. 1988).
21) Ogishi, T., Idoue, A., Hasegawa, T., Kato, T. and Suzuki, K.: Design and Implementation of Internet Performance Monitor with Realtime TCP Behavior Analysis, *IEICE Trans. Comm.*, Vol.E84-B, No.8 (2001).
22) Stevens, W.R.: *UNIX Network Programming*, Volume 1, Second Edition, *Networking APIs: Sockets and XTI*, Prentice Hall (1998).
23) http://www.antara.net.
24) Hargrove, P.H., Welcome, M. and Roman, E.: *MVIA 1.2*, Lawrence Berkeley National Laboratory (May 2001). http://www.nersc.gov/research/FTG/via.
25) PCI Special Interest Group (PCI SIG): PCI-X Specification Rev. 1.0a, http://www.pcicig.com.
26) InfiniBand (SM) Trade Association (IBTA): IBTA 1.0 Specifications (Oct. 2000). http://www.infiniband.org/.

(Received June 11, 2001)
(Accepted October 16, 2001)

Teruyuki Hasegawa received the B.E. and M.E. degrees of electrical engineering from Kyoto University, Japan, in 1991 and 1993 respectively. Since joining KDD in 1993, he has been working in the field of high speed communication protocol and ATM. He is currently a research engineer of Network Management System Lab. in KDDI R&D Laboratories Inc. He received Best Paper Award for Young Researchers of the National Convention of IPSJ in 1996, Best Paper Award of the National Convention of IPSJ in 1999, and Young Engineer Award of IEICE in 1999.

Tomohiko Ogishi received the B.E. degree of electrical engineering from the University of Tokyo in 1992. Since joining KDD in 1992, he has been working in the field of testing and monitoring of communicating systems. He is currently a research engineer of Network Management System Lab. in KDDI R&D Laboratories Inc. He received Best Paper Award for Young Researchers of the National Convention of IPSJ and Young Engineer Award of IEICE in 1998.

Toru Hasegawa received the B.E., the M.E. degrees in information engineering from Kyoto University, Japan, in 1982 and 1984, respectively. He received Dr. Informatics degree from the graduate school of informatics, Kyoto University in 2000. Since joining KDD in 1984, he has been working in the field of formal description technique (FDT) of communication protocols. From 1990 to 1991, he was a visiting researcher at Columbia University. His current interests are mobile computing and high-speed protocol. He is currently the Senior Manager of Moblie IP Network Lab. in KDDI R&D Laboratories Inc. He received IPSJ Convention Award in 1987.