Programmable Remote Traffic Monitoring Method Using Active Network Approach

Toru Hasegawa[1], Shigehiro Ano[1], Koji Nakao[1], and Fumito Kubota[2]

[1] KDDI R&D Laboratories Inc
2-1-15 Ohara Kamifukuoka-shi, Saitama 356-8502, Japan
{hasegawa,ano,nakao}@kddilabs.jp
[2] Communications Research Laboratory
4-2-1 Nukui Kitamachi, Koganei-shi, Tokyo 184-8795, Japan
kubota@crl.go.jp

Abstract. As the Internet has become an infrastructure for the global communication, a network failure and a quality degradation have become a serious problem. In order to solve the problem, a network monitoring system which monitors the traffic of Internet in real time is strongly desired. Traffic monitors which collect the statistics from captured packets play a key roll in the system; however, they are not flexible enough for being used in the rapidly changing Internet. The traditional approach such that a new traffic monitor is developed for a new requirement results in a long turn around time of the development. Therefore, we have proposed a flexible network monitoring system which consists of programmable traffic monitors. Traffic monitors are made programmable by introducing active network techniques; therefore, we call the network monitoring system as the active monitor network. This paper describes the implementation and evaluation of the active monitor network.

1 Introduction

As the Internet has become an infrastructure for the global communication, a network failure and a quality degradation have become a serious problem. In order to solve the problem, a network monitoring system is desired in order to monitor the quality and traffic of Internet in real time. Traffic monitors are key elements of the network motoring system. They are real time measurement tools which collect the statistics of the traffic from packets captured from a tapped link. Many traffic monitors are used for various purposes. MRTG (Multi Router Traffic Grapher) [1], which collects the number and bytes of IP packets from an MIB (Management Information Base) of a router, is used to detect congested links of an IP network. NetFlow [2] and NeTraMet [3], which collect the number and bytes of IP packets and TCP packets for flows, are used to monitor a traffic amount of an individual user. Recently, new traffic monitors [4], which collect TCP level quality statistics, are used to monitor a quality provided to an individual user.

The traffic monitors are useful to monitor an IP network; however, they are not flexible enough for being used in the rapidly changing Internet. The

I.W. Marshall, S. Nettles, and N. Wakamiya (Eds.): IWAN 2001, LNCS 2207, pp. 49–64, 2001.

traditional approach such that a new traffic monitor is developed for a new requirement results in a long turn around time of the development. Although new network monitoring applications such as a DOS (Denial of Service) attacker detection and a real time traffic quality monitor are emerging, the traffic monitors for them cannot be developed in time for network operatorsf requirements. Besides, traditional traffic monitors work stand alone; therefore, it is difficult to develop a network monitoring system which monitors a whole IP network in real time using traffic monitors distributed over an IP network.

In order to achieve a flexible network monitoring system, the introduction of active network approach [5] is very promising. We propose an active monitor network which consists of programmable traffic monitors and a manager. We call a programmable traffic monitor as an active monitor. A manager can dynamically load an analysis program which analyzes packets captured from a tapped physical link. This programmability achieves a flexible network monitoring system.

On the contrary, there has been much research [6,7,8,9,10] on introducing the active network into network management, of which network monitoring is an important role. In the above active network management systems such as SmartPacket [6], NetScript [7], and ANCORS [10], a management node and an agent node are made programmable. The programmability achieves so an intelligent agent behavior that MIB information is automatically checked. However, the active network management systems cannot make a network monitoring node, i.e., a traffic monitor, programmable. Although a program which runs on an agent analyzes the MIB information, it cannot analyze captured packet themselves. Recently, an active monitoring systems have been proposed [11], called as ANMOS Monitoring Tool. The active monitoring system focuses on monitoring an active network itself in order to know how an active network behaves. On the contrary, our active monitor focuses on traditional connectionless networks such as the Internet. Besides, our objective is to make a traffic monitor programmable so that network operators can collect traffic information which could not be collected unless parameters and sequences of captured packets were analyzed. We believe that our paper is the first proposal of the introduction of active network into traffic monitors in the literature.

In this paper, we propose an active monitor network and discuss the implementation and evaluation. In section 2, we describe the overview of the active monitor network. In section 3, we describe the implementation overview. In section 4, we describe the experiment results using the active monitor network. In section 5, we discuss the active monitor network approach.

2 Overview of Active Monitor Network

An active monitor network, as shown in Fig. 1, is a monitoring network which monitors an IP network consisting of routers and physical links. It consists of active monitors and a manager. An active monitor is a programmable traffic monitor, and consists of a platform and an analysis program which is remotely

loaded from a manger. An analysis program captures packets from a tapped physical link and analyzes the captured packets. The analysis results are stored in a result storage. A manager is also programmable. A monitoring application program manages active monitors, and runs on a manager platform. The application program remotely gets a result of an analysis program, and knows quality and traffic of a whole monitored IP network.

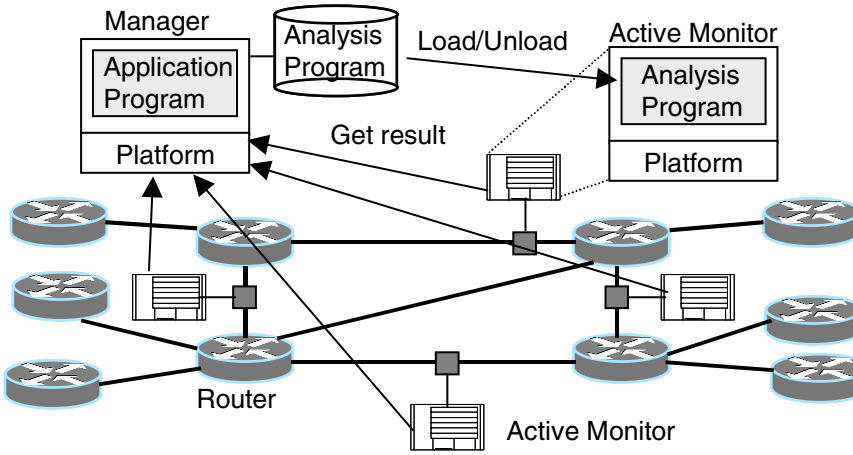

Fig. 1. Active Monitor Network.

2.1 Programmability of Active Monitor

Programmability of active monitor is achieved in the following way: An analysis program runs on a platform as shown in Fig.2. It is loaded beforehand from a manager to an active monitor via a network at any time without the monitor's being stopped. An analysis program corresponds to a programmable switch [12] in an active network. A standard programming language such as Java and C is used for writing an analysis program.

The platform provides an execution environment to analysis programs. It provides the following functions:

- Execution (Interpretation) of analysis program
- Load of analysis program
- Unload of analysis program
- Message communication to a manager
- Packet capture / filter

2.2 Programmability of Manager

A manager is also made programmable. A manager executes a network monitoring application program (application program in Fig. 1 and Fig.2). An application program controls active monitors located at many places in an IP network.

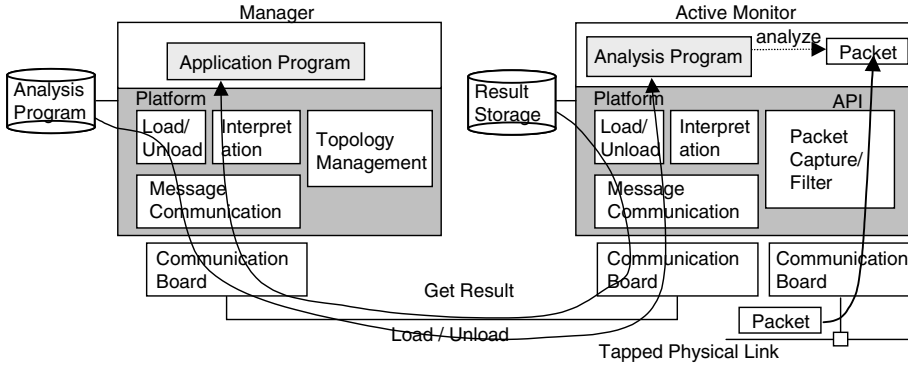

Fig. 2. Manager and Active Monitor.

The structure of manager is similar to an active monitor. It consists of an application program and a platform. The platform provides the load / unload, execution and message communication functions as the active monitor platform does. The platform also provides the topology management function, and it is described in section 2.4.

2.3 Communication between Manager and Active Monitor

The relationship of a manager and an active monitor is the same as that of a manager and an agent of traditional network management methods such as OSI management and SNMP based Internet management. An analysis program executed on an active monitor analyzes captured packets and stores an analysis result in a result storage like an MIB of SNMP. A manager gets an analysis result from an active monitor's result storage using client-server style communication. Three client-server style message exchanges are defined for the three operations: load and unload of analysis program and get result.

2.4 Network Topology

The manager platform provides an application program with a topology information of a monitored IP network and an active monitor network. The topology information is a list of pairs of link identifiers from the following two view points: First, a link is identified by source and destination router IP addresses in a monitored IP network. Second, the same link is assigned a unique identifier, which consists of an active monitor IP address and an identifier in the active monitor, within an active monitor network. The topology information is used for many purposes by an application program.

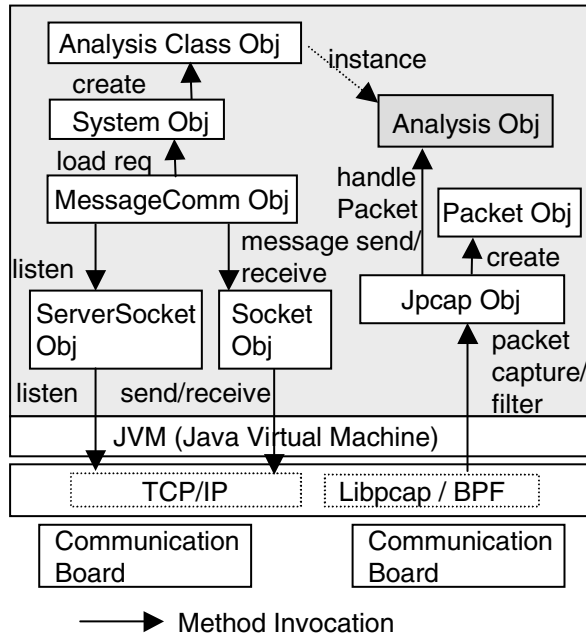

Fig. 3. Object Structure of Active Monitor.

3 Implementation of Active Monitor Network

An active monitor and a manager have been implemented using Java as the software which runs on standard PCs (Personal Computers) and workstations. Java is adopted as a programming language for writing an analysis program because of high productivity and portability. Currently, the developed software runs on Solaris 2.7/2.8 and Solaris 2.7/2.8 for x86 operating systems.

3.1 Active Monitor

(1) Program Structure

As shown in Fig. 3, an active monitor consists of Java objects (instances of Java classes) which run on the JVM (Java Virtual Machine). An Analysis object corresponds to an analysis program of section 2. The load and unload of an Analysis object is achieved by the creation and destroy of the class object of the Analysis class. The other objects constitute a platform. Individual functions of the platform are implemented as Java classes, and are executed as Java objects. A System object manages the platform and controls the other objects. A MessageComm object provides message communication using the two standard Java objects : ServerSocket and Socket objects. A ServerSocket object is used to

listen a new TCP connection, and a Socket object is used to send and receive messages using TCP/IP. A Jpcap object is a public domain Java object [13], and provides the packet capture and packet filter. A Jpcap object uses a libpcap and BPF (Berkley Packet Filter) programs which are outside the JVM for the packet capture and filter, respectively. Libpcap program is a public domain program which captures IP packets from a communication board. BPF is a public domain packet filter program. When a Jpcap object receives a packet from the libpcap/BPF program, it creates a Packet object.

(2) Analysis Class

When users need a new monitoring application, they write a class as a subclass of the SuperAnalysis class which provides a framework of packet analysis. They need overwrite three methods: the init, handlePacket and getResult methods. The init method is used to initialize an Analysis object. The handlePacket method is a main method. It is invoked by a Jpcap object when a packet is captured from a tapped link. The packet analysis is written in this method. The getResult method is used to send back an analysis result using a MesseageComm object.

An example of Analysis class, ExAnalysis, is shown in Fig. 4. This class calculates how many UDP packets whose source IP and destination IP addresses are specified by a packet filter condition (aFilter in Fig.4) are transferred on a tapped physical link.

When an ExAnalysis object is created, the init method is invoked by a System object with linkId, srcIPaddr and aFilter arguments. The arguments are sent from an application object of a manager. The aFilter is a BPF filter condition which specifies UDP packets of the specified source and destination IP addresses. Then, the run method is invoked, and it starts a Jpcap object assigning a thread. When the Jpcap object finds a packet which matches the filter condition specified by the aFilter, it invokes the handlePacket method. The handlePacket method further checks whether the destination port number of the UDP packet is the same as the destination port number of the targetPort instance variable. This procedure is infinitely repeated by the time when the quit method is invoked.

When an application object of the manager sends a get result request, a MessageComm object calls the getResult method. The getResult method checks whether the detected UDP packet number is larger than the threshold specified by the threshold. Then, it returns the result as a string to the MessageComm object, and the MessageComm object sends back the result to the manager.

(3) API (Application Programming Interface)

There is no constraint when writing Analysis classes. Users can use any methods provided by Java standard classes. The classes of Jpcap object provide analysis methods at MAC (Media Access Control), IP, UDP and TCP levels. The classes provide methods which get and set protocol parameters of the above protocols. On the contrary, the analysis of the protocols which upper than TCP, such as SMTP and WWW, is written in the handlePacket method using Java by users themselves.

```
public class ExAnalysis extends SuperAnalysis {
  private int packNum = 0; //  detected UDP packet no.
  private int threshold = 100; // threshold
  private int targetPort = 0; // destination port
  public void init (int linkID,  String srcIPaddr, String aFilter, int dstPort, …)
{
    targetPort = dstPort
    super.init (linkID, srcIPAddr, aFilter, ….);
    ….. }
  public void handlePacket (UDPPacket packet) {
    if (packet.dst_port == targetPort) {
      packNum++; // count up
     ….. }  }
  public String getResult ()  {
    String result; // Result storage
    If (packNum > threshold) { result = “TRUE”;  }
    else { result = “FALSE”; }
    return (result);   }
  public void run ()  {
    aJpcapObj.loopPacket (-1,this); }
  public void quit () {
    ….. }   }
```

Fig. 4. Example Analysis Class.

3.2 Manager

(1) Program Structure

The program structure of manager is shown in Fig. 5. The abstract class SuperApplication is provided for writing a network monitoring application. The platform consists of Java classes such as System, MessageComm, ServerSocket, Socket classes which are similar to those of the active monitor. Besides, the classes for topology management is provided. The class files of Analysis classes are stored as files.

(2) Application Class

A network monitoring application is written as a subclass of the SuperApplication class, and its object corresponds to the Application object in Fig. 5. A typical procedure of an Application object is as follows: First, the Application object loads a class file of an Analysis class to active monitors. The Analysis class is written beforehand and is compiled using a Java compiler to the class file by users. Second, the Application object gets analysis results from the loaded Analysis objects, and analyzes the results from the network wide view point. Finally, the Application object unloads the loaded Analysis objects at the active monitors.

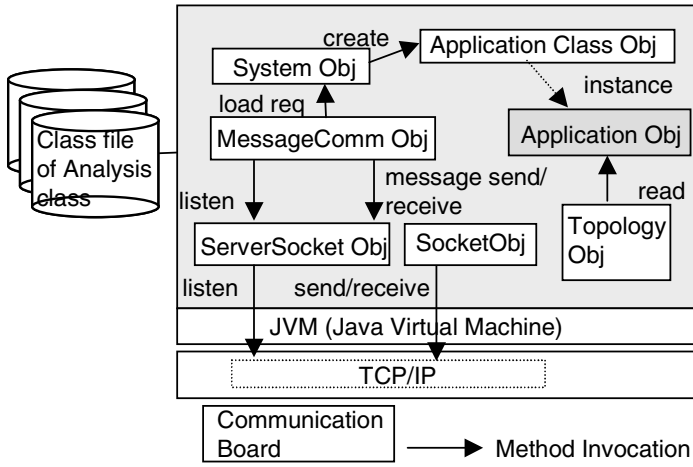

Fig. 5. Object Structure of Manager.

(3) Topology Management

A Topology object is provided so that users can write a method in consideration of locations of active monitors and a topology of a monitored IP network. The Topology object is a table of Link objects each of which represents a link. The link object contains source and destination router IP addresses, a bandwidth and an active monitor identification which consists of an active monitor IP address and a link identifier. The link identifier is assigned uniquely in the active monitor. Figure 6 and Table 1 show a monitored IP network and a table maintained by a Topology object, respectively. The monitored network consists of three routers whose IP addresses are 133.128.10.1, 133.128.12.1 and 133.129.11.1. Three active monitors are located to tap the all links among the routers. Each line of Table 1 corresponds to a Link object.

Many basic methods are provided to get the following basic information pieces from a Link object : an active monitor identifier, source and destination router IP addresses, a bandwidth of a link, a total Link object number, a Link object next to the currently accessed Link object and so on. In addition to the basic methods, many methods are provided to analyze a network topology of a monitored IP network. For example, the SPFmake method creates a shortest path first tree whose root is a specified. Bandwidths of a Topology object corresponds to metric of OSPF (Open Shortest Path First) routing protocol. If a monitored IP network uses OSPF as a routing protocol, the created shortest path tree can be used to know a route on which a captured IP packet is transferred. The MaxHop method calculates a maximum hop (link) number from a specified source router to a specified destination router.

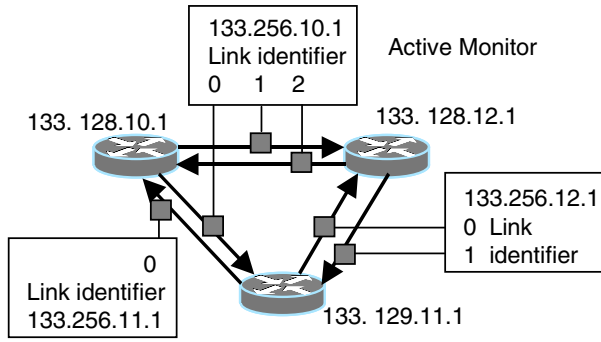

Fig. 6. Example Monitored IP Network.

Table 1. Topology Object.

active monitor IP address	Link identifier	Bandwidth (Mbps)	Source router IP address	Destination router IP address
133.256.10.1	0	156	133.128.10.1	133.129.11.1
133.256.10.1	1	100	133.128.10.1	133.128.12.1
133.256.10.1	2	100	133.128.12.1	133.128.10.1
133.256.12.1	0	10	133.129.11.1	133.128.12.1
133.256.12.1	1	10	133.128.12.1	133.129.11.1
133.256.11.1	0	156	133.129.11.1	133.128.10.1

3.3 Message Communication

The three operations (load, unload and get result) are achieved by an exchange of a request and a response using TCP/IP. MessageComm objects of a manager and an active monitor provide the communication. Figure 7 shows a typical example of message sequences between a manager and an active monitor. First, a manager sends a class file of Analysis object to an active monitor (load request at (i) of Fig. 7). After receiving the request, an active monitor starts executing the object and sends back the response to the manager. At some time goes, the manager sends a request to get an analysis result from the result storage of the active monitor (get request at (ii) of Fig. 7).Get requests can be set at any time and any times like a get request of SNMP. Finally, the manager sends an unload request to the active monitor (unload request at (iii) of Fig. 7). When receiving the request, the active monitor stops the execution of the Analysis object and unloads it.

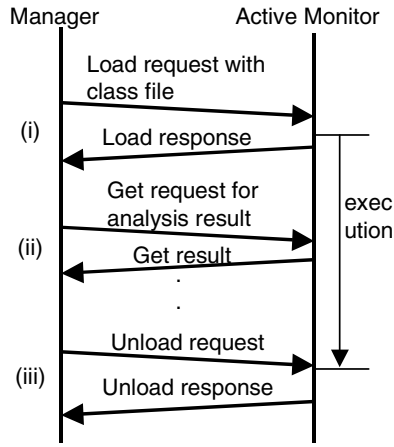

Fig. 7. Example Communication Sequence.

4 Network Monitoring Experiments

In order to evaluate the developed active monitor network, we have performed the two experiments. First, we have developed a DOS attacker detection application writing Analysis and Application classes, and have had experiments over a test bed network. Second, we have preliminarily measured the performance of active monitor.

4.1 DOS Attacker Detection Application Development

(1) DOS Attacker Detection Problem

Recently, DOS attacks such as ICMP Flood and TCP Flood are becoming a serious problem which degrades the Internet security. Usually, source IP addresses of DOS packets are forged (spoofed); therefore, a DOS attacker who sends DOS packets cannot be identified using source IP addresses of the packets. Due to this, the method of detecting DOS attackers is becoming an important problem [14] for a network monitoring system.

We have developed a DOS attacker detection application. The DOSApplication and DOSfilter classes are written. They correspond to the Application and Analysis classes, respectively. A DOSfilter object is used to detect DOS packets. As for DOS packets, the destination IP address is a target host of a DOS attack. Source IP addresses are forged; however, usually the addresses are in some network number specified by an IP address and an address prefix. Therefore, a DOSfilter object detects a DOS packet by filtering captured packets with the filter condition of the above destination IP address and the source IP network

number. The DOSfileter class is similar to the ExAnnalysis class of section 3.1, and returns TRUE or FALSE when a get result is requested.

The DOSApplication class detects a link from which an attacker sends DOS packets. The algorithm of the class is explained using an experiment network configuration shown in Fig. 8. The network consists of 5 routers and 7 LANs (Local Area Networks). IP addresses of routers and network numbers of routers are shown in Fig. 8. For example, IP1 is an IP address of a router, and NW1 is a network number of a LAN. The detection algorithm is designed on the following assumptions: First, all links are tapped by active monitors. Second, the IP network uses OSPF as a routing protocol, and it comprises a single area of OSPF. Besides, a Topology object maintains the same topology information as that of OSPF. Third, all links between two routers are symmetric and the bandwidths of both directions are the same.

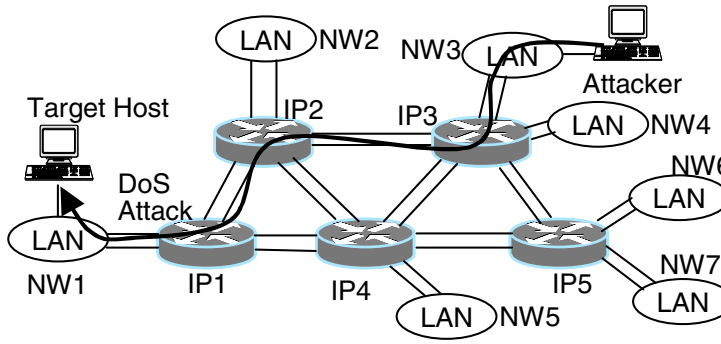

Fig. 8. Experiment Network Configuration.

(2) Simple Algorithm

Since all links are monitored, the simplest algorithm is to load DOSfileter objects to all active monitors. Getting the results whether a DOS packet is detected, a manager knows the links on which DOS packets are transferred, and finally knows the nearest link to the attacker by combining the links. This algorithm is straight forward and requires many message exchanges between a manager and active monitors. Since the network of Fig. 8 consists of 28 links, 28 message exchanges happen.

(3) Advanced Algorithm to Reduce Message Number

We have designed an advanced algorithm using the topology information in order to reduce the message exchange number. Since the network uses OSPF and the bandwidths of the both direction links are the same, an IP packet takes the same route both from a source to a destination and from a destination to a source. Therefore, all packets sent to a target host of LAN NW1 are transferred on routes of the SPF (Shortest Path First) tree whose route is the LAN NW1, as

shown in Fig. 9. Using this tree, DOS attackers are detected using less messages than the simple algorithm. The DOSApplication detects DOS attackers in the following way:

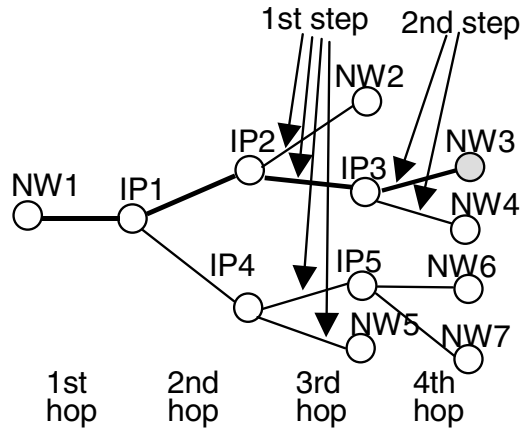

Fig. 9. Detection Using SPF Tree.

(i) The DOSApplication object calculates the SPF tree by invoking a method of a Topology object. Then, it calculates a maximum hop number among all routes from LANs to LAN NW1. In Fig. 9, the maximum hop number is 4.
(ii) It calculates the half of the maximum hop number so that DOSFileter objects are loaded to the next links to the middle routers of the SPF tree. In Fig.3, the links of the 3rd hops are the links.
(iii) It loads DOSFilter objects to active monitors which tap the links of the calculated hop number. It then gets the analysis results. When the DOSApplication object gets links on which DOS packets are detected, it loads DOSFilter objects to active monitors of the next hop links to the detected links. In Fig. 9, the link between IP3 and NW3 and that between IP3 and NW4 are the links. This procedure is repeated until when all routes of the DOS packets are fixed.

In Fig. 9, when the 4th hop links are checked, the algorithm finishes and detects an DOS attacker at the link between IP3 and NW3. In this example, only 6 message exchanges happen although 28 message exchanges happen when using the simple algorithm.

We have actually performed the experiment using the network configuration shown in Fig. 8. The DOS attacker can be detected in about just 2 seconds.

4.2 Performance Evaluation

The performance of packet capture and filter has been evaluated in the following way: An active monitor runs on a PC with a 1.3GHz Pentium III CPU and 512 Mbyte memory. The PC taps a FastEthernet link and a LAN tester (IXIA) sends

packets at the fix rate changing packet sizes. The ExAnalysis object of section 3 runs and it filters the packets using source and destination IP addresses. The result is shown in Table 2. Each column shows the packet capture speeds in the two forms: the captured packet number per second and the capture throughput.

Table 2. Performance of Active Monitor.

Packet Size Bytes	64	128	256	512	1024
Capture Speed Packets / sec	7400	6100	4010	2470	1240
Capture Throughput Mbit/s	3.78	6.25	8.21	10.12	10.16

5 Discussion

5.1 Usefulness of Active Monitor Network

(1) Flexibility and Programmability

The introduction of active network techniques into traffic monitors and a network monitoring system is quite successful. The programmability of traffic monitor is so useful that the DOS attacker detection application can be implemented in just two weeks by one Java programmer. The short turn around time of a network monitoring application development is one of the most important advantages of active monitor networks. If the application were developed from the scratch, it would take more than half a year because the platform development took half a year. The good productivity is achieved by the following aspects: First, the active monitor platform provides abundant methods for analyzing captured packets such as the packet filter and the get/set of protocol parameters. This improves the productivity of Analysis class. For example, an ExAnalysis class is about just 80 line long. Second, the client-server style communication similar to SNMP is simple, it is easy for programmers to write a network monitoring application program. Third, Java itself provides the high productivity.

(2) Topology Information

Topology information is quite useful for programmers to write an Application object which analyzes a monitored network in consideration of topology. As for the DOS attacker detection application, the topology information is successfully used to reduce the number of links on which active monitors must monitor. Besides, the topology information is expected to be used by many network monitoring applications. For example, it will be used to calculate a network performance on each route.

(3) Performance

The performance of active monitor is not so high. However, the performance is enough for many network monitoring applications. For example, the DOS attacker detection application of this paper works correctly as long as some portion of packets is captured. In order to improve the performance, a performance bottleneck object, i.e., a Jpcap object, need be improved. As shown in Table 2, the packet capture number per second decreases as the packet size increases. This decrease is due to the garbage collection of JVM. Since the Jpcap object uses more memory for a larger packet than for a smaller packet, garbage collections more often happen for the larger packet size. Therefore, we plan to rewrite the Jpcap object in other programming languages such as C or assembly languages. This reduces the garbage collection times, and improves the performance.

5.2 Comparison with Related Work

(1) Traditional Traffic Monitors

So far, many traffic monitors [1,2,3,4] were developed. The traffic monitors collect the statistics from captured packets. None of the monitors are programmable; in other words, our active monitor is the first programmable traffic monitor in the literature. Besides, by writing programs, our active monitor can provide any function provided by other traffic monitors.

(2) Active Network Management System

Many active network management systems [6,7,8,9,10] were developed. These systems make traditional manager-agent based network management systems programmable. The systems make an agent storing MIB information programmable. For example, a packet of SmartPacket [6] contains a program for detecting equipment errors or recovering from a failure, and the program is executed on the agent, which reduces the communication between a manger and an agent. However, these systems just focus on intelligently handling of MIB. They do not provide the functions for handling the traffic statistics which are not stored in the MIB.

ANCORS [10] is an adaptable network control and reporting system which merges network management and distributed simulation. The system provides user-definable network monitoring capabilities. Although the analysis of monitores result is programmable, the system just uses the existing traffic information such as MIB and RMON-MIB inforamtion.

ANMOS monitoring tool [11] is the first active network management system which focuses on network monitoring. However, this tool focuses on monitoring an active network itself in order to know how an active network behaves. Besides, it does not make traffic monitors programmable. On the contrary, our active monitor makes a traffic monitor programmable so that network operators can collect traffic information which could not be gotten unless parameters and sequences of captured packets were analyzed.

(3) Active Network Platform

An Analysis program is executed every when a captured packet is received. This mechanism is similar to capsule based active networks where a program

in a capsule is executed when a capsule is received by an active node. PLANet (Packet Language for Active Networks) [15], ANTS [16] and SC Engine [17] are examples of capsule based active networks. The implementation of our active monitor network is similar to ANTS; however, the objectives are different. The other capsule based active networks make routers programmable; however, our active monitor network makes traffic monitors programmable.

6 Conclusion

In order to achieve a flexible network monitoring system, we propose the introduction of active network approach into traffic monitors. We have developed an active monitor network which consists of programmable traffic monitors and a manager. We have also developed a DOS attacker detection application by writing an analysis program and an application program. The following results are made clear from the above developments.

- The introduction of active network approach into traffic monitors is useful to achieve a flexible network monitoring system. The programmability of traffic monitors makes the productivity of a network monitoring application high. For example, the DOS attacker detection application has been developed in two weeks.
- The introduction of topology information of a monitored IP network into a network monitoring system is useful to make a network monitoring system intelligent. For example. The DOS attacker detection application decreases the message number between a manager and active monitors using a shortest path first tree calculated from the topology information.

References

1. T. Oetiker, "Multi Router Traffic Grapher (MRTG) Homepage," `http://www.mrtg.org/`.
2. Cisco Systems Inc., "NetFlow FlowCollector Homepage," `http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/nfc/`.
3. N. Brownlee, "NeTraMet Homepage," `http://www.auckland.ac.nz/net/NeTraMet`.
4. T. Kato, et. al., "Performance Monitor for TCP/IP Traffic Analyzing Application Level Performance," Proceeding of ICCC'99, vol.2, pp.114-121, September 1999.
5. D. Tennenhouse, et. al., "A Survey of Active Network Research," IEEE Comm. Mag., vol.35, no.1, January 1997.
6. B. Schwartz, et. al, "Smart Packets for Active Networks," Proceeding of OPENARCH'99, March 1999.
7. Y. Yemini and S. da Silava, "Towards Programmable Networks," Proceeding of IFIF/IEEE International Workshop on Distributed Systems: Operations and Management, October 1996.
8. G. Pavlow, et. al, "The OSIMIS Platform: Making OSI Management Simple," Proceeding of ISINM'95, 1995.

9. R. Kawamura and R. Stadler, "Active Distributed Management," IEEE Communication Magazine, pp. 114-120, April 2000.
10. L. Ricciulli and P. Porras, "An Adaptable Network COntrol and Reporting System (ANCORS)," Proceeding of the 6th IFIP/IEEE International Symposium on Integrated Network Management, May 1999.
11. M. Ott, et. al., "Looking Inside an Active Network: The ANMOS Monitoring Tool," Proceeding of IWAN 2000, October 2000.
12. D. Alexander, et. al., "The Switch Ware Active Network Architecture," IEEE Network Mag., June 1998.
13. `http://www.goto.info.waseda.ac.jp/~fujii/jpcap/index.html`.
14. R. Stone, "CenterTrack: An IP Overlay Network for Tracking DoS Floods," NANOG, October 1999.
15. M. Hicks, et. al., "PLANet: An Active Internetwork," Proceeding of INFOCOM'99, March 1999.
16. D. Wetherall, et. al., "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," IEEE OPENARCH'98, April 1998.
17. F. Kubota, et. al., "Congestion Management based on Routing Functions over Active Internetwork System," Proceeding of APNOMS 2000, October 2000.