2-6 Log Management System Based on Distributed Database Using P2P Network

KAMIO Masakazu, ISHIDA Tsunetake, and HAKODA Takahisa

It is necessary to investigate the phenomena in order to perform the countermeasures to illegal access on computer network, and it is aimed against the logs from network nodes. Now these investigations and analysis greatly depend on network administrators with advanced technology. But the number of the nodes is increasing along with the computer network expansion, and the investigations are becoming more difficult. So we are researching and developing the log management system based on the distributed database, which enables efficient log management and analysis under the large-scale network environment.

Keywords
Agent technology, Distributed computing, Peer-to-Peer, Management

1 Introduction

The growing development of computer networks has been accompanied by an increasing number of victims of illegal access and computer viruses or worms[1]. To prevent these incidents, it has become commonplace to use firewall and "vaccine" software programs. More recently, security measures have included network-level monitoring or alliances with security equipment, using intrusion detection systems (IDS) and intrusion prevention systems (IPS). However, given that the number of victims is still on the rise, it is clear that these measures are not fully effective in preventing such incidents. In addition to taking protective measures against intrusions, it is necessary to check continuously for anomalous events and to respond accordingly. However, the greater use of computer networks has led to an increase in the number of nodes and in the complexity of network configurations. For large organizations operating in geographically dispersed locations, it is becoming difficult to check the various nodes that make up the computer network. This checking and analysis is now largely dependent on the advanced skills of network administrators. However, it is not always possible to hire sufficiently capable administrators. Moreover, if a computer network consists of geographically dispersed sub-networks, a different administrator must be assigned to each location.

These problems have recently been addressed by technology developed for the efficient management of multivendor computer networks, such as Web Based Enterprise Management (WBEM)[2][3] and Application Resource Management (ARM)[4]. This technology is effective in managing various machines within an organization, but this highly integrated mechanism may not be used for central management of networks operated by different organizations. This is mainly because the respective organizations are unable to cede network management authority to third parties.

However, we believe that it remains possible to establish loosely knit coordinated relationships in which organizations can exchange the appropriate messages or inquiries upon

detection of an anomaly. We have been working on R&D of a log collection/management system that consists of multiple, disparately located servers that feature log collection/management and anomaly detection functions. Upon detection of an anomaly at one server, the other servers will search for and provide related information and log data[5]-[7].

In this paper, we will describe the concept of a log collection/management system that coordinates log collection/management servers. We will also describe a statistical method for detecting anomalies in collected logs, a fast pattern matching method using automatons, and a P2P (peer-to-peer) protocol for flexibly coordinating log collection/management servers.

2 Studies on configuration

2.1 Size and configuration of networks

This log collection/management system is intended to adapt flexibly to a wide range of networks: from a small network featuring a single sub-network consisting of less than 20 nodes to a large network of sub-networks operated by different organizations. First, we focused on various network sizes and suitable configurations for each.

Table 1 *Classification by size of network*

	Small	Medium	Large
Number of nodes to be managed	＜20	20－100	100＜
Number of networks	1	1＜	1＜
Number of operating organizations	1	1	1＜

First, we classified computer networks based on the sizes shown in Table 1 and investigated system configurations suitable for each environment.

A small computer network is a single network consisting of less than 20 nodes operated by a single organization. Since the amount of collected logs is small and log transmission traffic is low in such an environment, log collection and management is appropriately conducted by one server.

A medium-sized computer network consists of several small sub-networks operated by a single organization. If a single server collects logs in such an environment, traffic will increase with increasing proximity to the server, which may place considerable load on the network. Moreover, with an increase in the amount of logs concentrated in a single server, storage space and processing capacity may become insufficient. To prevent this and related problems, the log collection/management function should be distributed among the sub-networks, with central control of their operations. In other words, a hierarchal system configuration is most appropriate for a medium-sized network.

A large computer network consists of several small or medium-sized sub-networks operated by different organizations. In such an environment, each organization operating a sub-network must collect and manage logs on its own. The centrally controlled configuration used in a medium-sized network is not suitable for a large network. However, when an anomaly occurs in a sub-network, the search for relevant information must extend to all sub-networks, as all of these networks are linked. To meet these requirements, a multi-agent system configuration is required in which each log collection/management server collects logs from its own sub-network nodes, coordinating with the other servers as necessary.

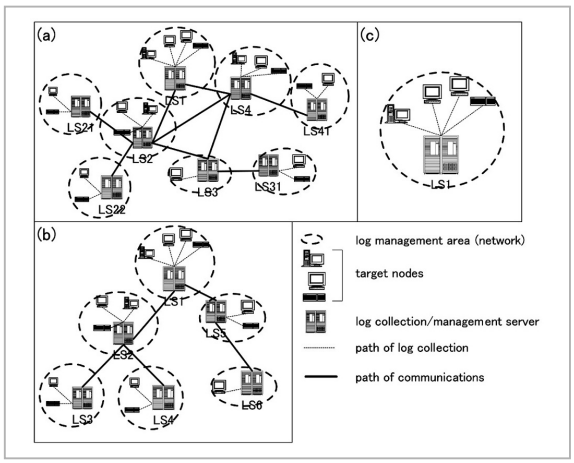

Fig. 1 *Suitable configuration based on network size*

In short, log collection/management systems must operate according to the network size and configuration. However, it is possible to implement the above-mentioned single-server and hierarchal system configurations as variations of the multi-agent system configuration shown in Fig.1.

Figure 1 (a) shows a multi-agent system configuration in which log collection/management servers coordinate at a single level. In Fig.1 (b), log servers are placed within a tree structure. In this way, a layered system configuration for a medium-sized computer network can be implemented as a variation of the multi-agent system configuration. As shown in Fig.1 (c), a single-server configuration can be implemented in the form of a sub-network that would constitute a part of the multi-agent system.

If log collection and management is conducted across several organizations in a large network environment, the functions of each log server must be adjusted according to its relationship with others. For example, a log server may provide all available log information to a node within the same organization, while limited to returning only information of presence or absence of the relevant logs to other organizations. In this case, the multi-agent system enables the establishment of policies governing responses to requests from individual log servers, without the need to adjust the network configuration. As a result, a log collection/management system can be designed more flexibly.

The conclusions described above show that, based on a multi-agent system that coordinates independent log collection/management servers, it is possible to construct a log collection/management system adaptable to networks of various sizes and configurations. These log servers can switch autonomously among policies—according to the internal or external nature of communications, for example. These servers can thus coordinate effectively regardless of the network configuration.

2.2 Log collection/management servers

We then proceeded to a study of servers used for log collection and management.

A log collection/management server must collect logs from various nodes such as application servers, routers, switching hubs, firewalls, and IDSs. Formats of collected logs also vary: the “system logs (syslogs)” commonly used by UNIX systems, the “Event Logs” used by WindowsNT, the “SNMP (Simple Network Management Protocol)” often used by routers and switching hubs, and email used by IDSs to notify administrators. In addition, some nodes need to employ original log formats. Obviously, a single log sever cannot handle all of these formats; the range of receivable log formats or protocols must be expandable. Two methods are available to extend this range: (1) definition of new log formats or protocols for log collection/management, and installation of agents at the appropriate nodes to collect these logs; and (2) as necessary, enabling the specification of additional log formats or protocols receivable on the log collection/management server side. With method (1), it is possible to accommodate new log formats or protocols by developing agents at the new nodes required to collect these logs, eliminating the need to make changes to the rest of the network. However, in some cases it is not possible to install additional programs (i.e., agents) on certain nodes. We thus decided to use method (2) to extend the range of collected log formats or protocols.

To help network administrators check and analyze logs, this log collection/management system must be able to check collected logs and detect anomalies automatically. Further, to collect and manage different log formats, this system needs be able to: (1) add checking functions according to log formats; and (2) check across logs and logs of multiple formats. We believe that function (1) will allow servers to provide flexible checking functions according to the managed log formats, and that function (2) will allow servers to detect anomalies more effectively by evaluating multiple events—for example, by comparing the detection results collected from different

nodes through IDSs and those of syslogs from servers.

This system makes it possible to extend the range of receivable logs as necessary. However, since log checking functions are dependent on receivable log types, it is unlikely that previously implemented log checking functions can be used to check newly added log formats. Therefore, when extending the range of receivable logs, it is also necessary to extend the system's log analysis functions.

As described above, log collection/management servers must allow for the easy addition or modification of functions. To this end, we designed a group of functional modules that together make up a log collection/management server. Figure 2 shows the layout of this structure.

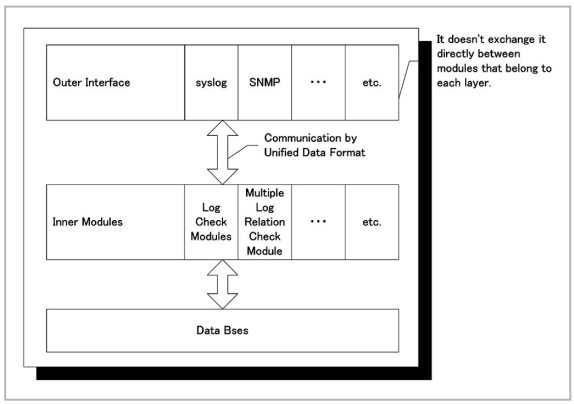

Fig.2 *Conceptual Diagram of log collection/management server configuration*

As shown in Fig.2, a log collection/management server consists of a number of functional modules. A strong dependent relationship between certain modules renders it difficult to add, delete, or modify functions flexibly. To reduce dependency and to ensure ease of addition, deletion, and modification of functions, we divided the server into layers and set up a common interface between the layers.

We designed the server to consist of three layers: (1) an external interface section in charge of external communications, such as reception of logs from other nodes, coordination among servers, and conversion of internal and external data formats (for communication with other nodes); (2) an internal processing section in charge of analyzing received logs, checking relations among accumulated data items, and generating messages to initiate coordination upon detection of anomalies; and (3) a concealed database section that features a hidden database for storing received logs and provides a common interface to upper modules.

3 Proposed system

3.1 Log collection/management server

Figure 3 shows a specific configuration of a log collection/management server. We will use this figure to describe how the server works.

The three-layered rectangle in the center of the figure represents the log collection/management server. Outside this log server, there are nodes to be managed for log collection, other log servers for the external interface, and a user interface for server management. As described above, a log server consists of the external interface, internal processing, and database sections (layers). These sections (layers) include functional modules of the log server. Arrows in the figure indicate the flow of log data from reception to storage in the database (Logs), data related to checking and analysis within the server (Internal messages), and data related to coordination with other servers (Cooperation messages).

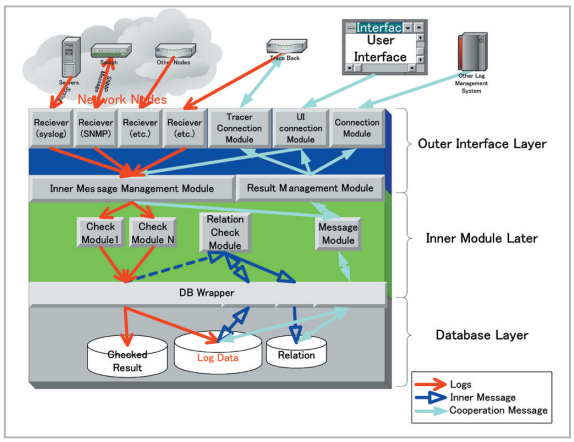

Fig.3 *Configuration of log collection/management server*

3.1.1 External interface section

The external interface section includes modules for reception of logs and coordination with other log servers and user interfaces. Each log reception module receives logs in a specific format and converts the format for use inside the server.

The internal message reception and result notification modules control the flow of data between the external interface and internal processing sections. This helps ensure the independence of these two sections.

3.1.2 Internal processing section

Received logs are passed to anomaly check modules in the internal processing section. After checking is finished, these logs and the check results are entered in the database (indicated by arrows labeled "Logs"). It is possible to set up a separate module for each log type. It is also possible to associate a log type with two or more detection methods. In this case, several types of check results are linked to a given log.

The anomaly check modules check received logs for anomalies before the logs are stored in the database. In addition, the relevant anomaly check module checks logs that were received at different times or from different nodes (indicated by arrows labeled "Internal messages"). These check results are monitored by the relevant modules. To check more extensively, requests for the relevant information are sent to the other log servers as necessary (indicated by arrows labeled "Cooperation messages").

3.1.3 Database section

The bottom layer of log collection/management is formed by a database that stores received logs and check results. To ensure that the database is scalable, between the internal processing section and database, we installed a DB management module that hides the database system and provides a common interface with the internal processing section.

3.2 Coordination functions

This is a multi-agent distributed system in which log collection/management servers act as agents. The system is designed to coordinate computer networks operated by different organizations. We devised a P2P-based coordination scheme because a P2P network enables easy installation of agents, regardless of the overall network structure.

In this scheme, log collection/management servers operate as follows. First, each log collection/management server selects several neighboring log servers and establishes communications. When a log collection/management server detects an anomaly through its collection and analysis of logs, it will issue notification and a request for the relevant information to the neighboring log collection/management servers in a peer-to-peer fashion. Upon reception of this message, the other log servers will check their stored logs, and any log data judged as related to the anomaly will be returned to the source (server). In addition, the servers will forward this message to all neighboring log collection/management servers except the log collection/management server that issued this message. This procedure is repeated to check logs throughout the network, as shown in Fig.4.

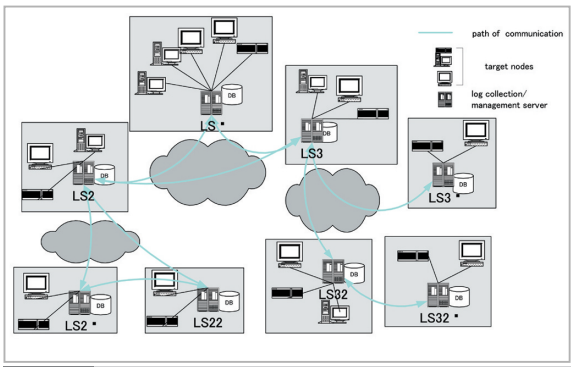

Fig.4 *Coordination of log collection/management servers*

In the figure, for example, LS1 communicates with LS2 and LS3, and LS2 communicates with LS1, LS21, and LS22. When LS1 sends a message to LS2, LS2 will forward this message to LS21 and LS22. In this way, LS1 can coordinate with LS21, LS22, and the other log servers with which LS1 does not have direct communication.

However, in a loop structure in which

neighboring log servers communicate with one another, message transmission overlaps, generating unnecessary traffic. In the case shown in Fig.4, for example, when LS21 and LS22 receive a message from LS1, each will send this message to the other. Gnutella[8], a typical P2P protocol, prevents an increase in traffic to a certain extent by discarding overlapping messages, but traffic still increases in proportion to the square of the number of nodes involved, placing pressure on the network bandwidth. Chord protocol[9], which employs P2P in distributed databases, prevents an increase in traffic by adjusting data locations and search procedures using the flexibility of the P2P network. However, this system requires a protocol that can broadcast messages (like Gnutella) throughout the system, as log locations are unknown in this system. Moreover, since Chord does not take bandwidth or route status into consideration, low-speed routes may be selected to send a given message. To coordinate among log collection/management servers, we devised a P2P protocol that is able to optimize communication routes among neighboring nodes. Fig.5 shows the route control method using the P2P protocol we have devised.

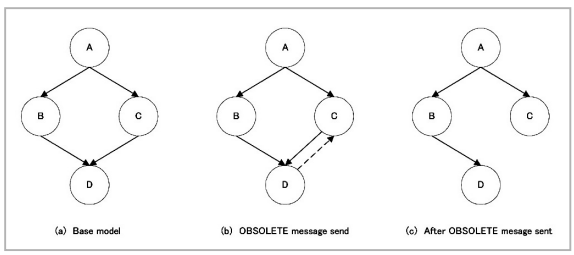

Fig.5 *Routing method*

In Fig.5, a node A communicates with neighboring nodes B and C; and B and C communicate with neighboring node D [Fig.5 (a)]. When A sends a message, D will receive this message from both B and C. In the Gnutella protocol, node D will discard one of the messages received from B and C, as described above. However, the generation and transmission of such overlapping messages is simply unnecessary. According to our routing method, if a node receives the same request message several times, this node will send a transmission stop request (OBSOLETE) to nodes sending the second or later transmission of this request message. Nodes that receive the OBSOLETE command will stop sending messages to the requesting node. For example, when D receives the same request message from B and C, D will send the OBSOLETE command to the node (C in this example) that sent the message later [Fig.5 (b)]. After receiving the OBSOLETE command, C will stop forwarding request messages from A to D [Fig.5 (c)]. This method prevents subsequent request message transmissions from causing unnecessary traffic. Before the OBSOLETE command is sent, a given message transmission results in $O(n^2)$ of traffic, as in the Gnutella protocol. After the OBSOLETE command is sent, traffic caused by a given transmission will decrease to $O(n)$.

A disadvantage to this method is seen in that transmissions may be interrupted when trouble occurs on a node that forwards messages, as shown in Fig.6.

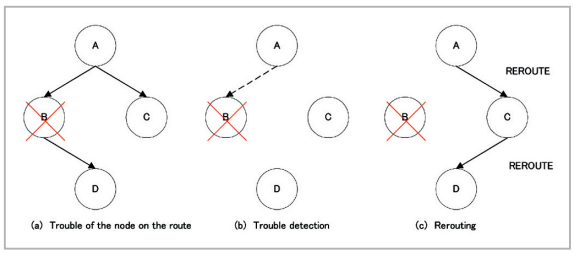

Fig.6 *Rerouting in case of trouble*

If trouble occurs at B after the route is established as shown in the previous figure, messages will not be sent to D. The detected trouble of the destination node can be solved by rerouting messages regardless of the earlier OBSOLETE command. When attempting to establish a connection with B, if A finds that B is unable to forward messages, then A will send an OBSOLETE cancel request and other messages (not having been forwarded by B) to C so that C can forward the messages to D.

The new protocol described above prevents the increase in traffic caused by conven-

tional P2P protocols such as Gnutella. This enables all servers in this log collection/management system to exchange information and receive results using the flexibility of the P2P network—in other words, it enables optimization of communication routes.

4 Log checking functions

This system is designed to conduct anomaly detection at each of its log collection/management servers. This system allows for the use of existing methods, such as detection by pattern matching, and detection based on traffic patterns collected through SNMP.

Aside from these methods, we devised a classification method based on the frequency of occurrence of words within logs and a high-speed pattern matching method using automatons. This section will describe these methods.

4.1 Classification based on the frequency of occurrence of words

This method involves sampling known logs and classifying them into certain categories, creating an "occurrence frequency matrix" that shows which words occur at what frequencies in each log category. This matrix can be used as a set of decision functions to classify newly received logs.

This method enables the creation of decision functions tailored to individual environments based on past data. This method thus offers a number of advantages over pattern matching: (1) the ability to detect anomalies when a log server receives unknown logs; and (2) the ability to create decision functions suited to individual environments based on sampled logs.

4.1.1 Creation of decision functions

An occurrence frequency matrix consists of words (in rows) and categories (in columns), and each element shows the frequency of occurrence of a word in each category. A matrix that corresponds to a group of words is called a corpus. Figure 7 shows the layout of a corpus.

To create a corpus and decision functions,

$$\begin{pmatrix} & \begin{pmatrix} cat_1 & cat_2 & cat_3 \end{pmatrix} \\ \begin{pmatrix} word_1 \\ word_2 \\ \cdot \\ \cdot \\ word_n \end{pmatrix} & \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ e_{n1} & \cdot & e_{n3} \end{bmatrix} \end{pmatrix}$$

Fig.7 *Layout of corpus*

the following steps are performed on the user side: (1) collection and classification of logs for use as original data; (2) morphological analysis of the classified log messages to extract words; (3) elimination of white spaces, punctuation marks, and character strings that consist only of numbers; (4) if a word extracted in Step (2) is new, an additional row is created to hold this word, and the total for the number of words (rows) is augmented by one. If an extracted word is an existing one, the existing category value is increased by one; (5) the single value indicated in each of the columns created in Step (4) is divided by the number of logs included in that category; and (6) each element (from the results of Step (5) is divided by the total for all elements in that row.

The resultant matrix is used as a set of decision functions.

4.1.2 Classifying logs

To classify a log, the user performs the following steps: (1) checks whether a word contained in the corpus occurs in the log—if it does, "1" is written to the location in the corpus corresponding to this word; if not, "0" is entered; (2) repeating Step 1, vectors of ones and zeros are created that correspond to the occurrence or non-occurrence of the words; (3) these vectors are multiplied by the decision function vectors; and (4) the element with the largest vector value as a result of the calcula-

tion in Step 3 is then selected. The number assigned to this element will correspond to the number of the category to which this log belongs.

4.2 Pattern matching by automatons

We devised this method to speed up anomaly detection based on log occurrence patterns. Through parallel operation of several automatons linked without interruption and based on the relationships among multiple logs, we tried to reduce the number of anomaly detection checks, and also, to verify several patterns to increase detection speed.

4.2.1 Configuration and operation of automatons

As shown in Fig.8, an automaton consists of several nodes, each representing specific states. Each node has regularly expressed conditions for a shift to the next state. A log collection/management server passes received logs sequentially to automatons for verification.

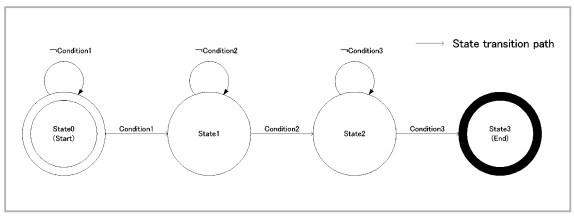

Fig.8 Configuration of automaton

An automaton checks logs as follows: (1) When receiving a log, the automaton will check whether the log fulfills the conditions for a shift to the next state. (2) If it does, the automaton will shift to the next state. (3) A shift to the final state means that an anomaly has been detected. In this case, the automaton will record the related log data and then return to its initial state. Logs are checked sequentially through repetition of these steps.

It is sometimes appropriate to presume that logs widely separated in time are not related. Therefore, we specified a time period after which logs are discarded. After this period has elapsed, the automaton returns to its initial state.

4.2.2 Parallel operation of automatons

The user prepares the necessary number of automatons according to the desired patterns to check; these automatons then operate in parallel.

Figure 9 shows the concept of automatons operating in parallel. This group of automatons operates as follows: (1) A log collection/management server passes a received log to the first automaton [Automaton (0) in Fig.9]. (2) This automaton checks the log, shifts its state accordingly, and sends the log to the next automaton. (3) For each log, Step 2 is performed by all automatons. (4) Each time a new log is received, Steps 1 through 3 are triggered.

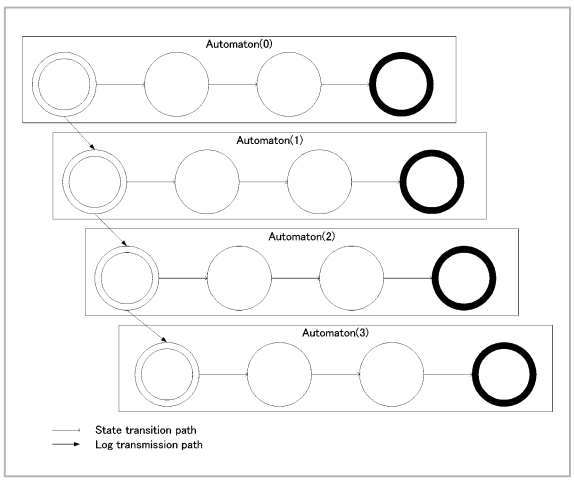

Fig.9 Parallel operation of automatons

These automatons are series-linked and every automaton is always in one of the designated states. Therefore, the amount of time required to check a log depends only on the number of automatons. Since these series-linked automatons operate in parallel, it is possible to handle as many logs as the number of automatons when logs are received in succession.

5 Implementation

Based on the concept described so far, we are in the process of prototyping a log collection/management system. Figure 10 shows the functions we have already implemented on a

module-by-module basis.

This prototype uses its reception modules (for syslog, SNMP, SMTP, and POP3) to receive syslog, SNMP, and mail logs, and runs anomaly detection routines for an each log before storing these logs in its database. At present, the prototype features only one detection routine based on the frequency of occurrence of words, as described above. However, the superordinate internal message reception module allows the user to specify the modules to be used next. In the future, it will be possible to tailor anomaly detection routines even more precisely by selecting the optimal module, according to the type of log received.

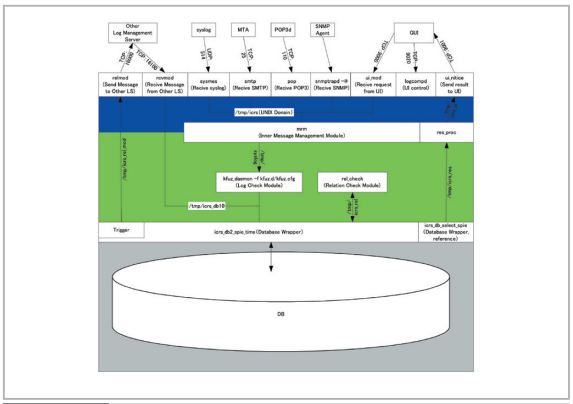

Fig. 10 *Overview of implementation*

Upon reception of a log, the relation check module conducts anomaly detection based on the reception state of several logs. In this anomaly detection, automatons perform the pattern matching described above.

If a log collection/management server detects an anomaly through any of its detection functions, the coordination trigger module located in the database wrapper will decide, based on the settings, whether or not to start coordination. If this module decides to start coordination, it sends a coordination message to the other log servers. Upon reception of this message, the other log servers will check their own databases, and if they find related log information, they will send it back to the server in which the anomaly was detected. In addition, they will forward this message to all neighboring log servers except the log server that issued this message. In this way, network-wide coordination is achieved through the protocol described in **3.2**, "Coordination functions".

6 Results and discussion

We checked the operation and evaluated the performance of the implemented functions. This section will briefly describe our test results.

We measured traffic when the new P2P protocol was employed, and compared this protocol with Gnutella based on the following prediction: before the OBSOLETE command is sent, message transmission will cause $O(n^2)$ of traffic (as in the case of Gnutella); after the OBSOLETE command is sent, traffic will decrease to $O(n)$, as the transmission route will then feature a theoretical tree structure.

Figure 11 shows the results of traffic measurement. As we predicted, prior to route control message transmission results in nearly the same volume of traffic as with Gnutella. After the OBSOLETE command is sent, traffic simply increases in proportion to the number of nodes.

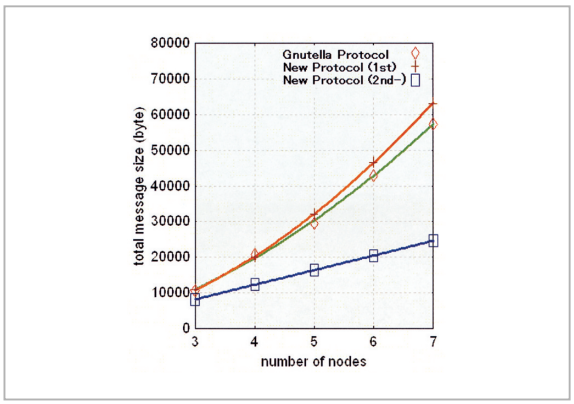

Fig. 11 *Results of traffic measurement*

We evaluated the accuracy of anomaly detection using the classification method based on the frequency of occurrence of words. We used the "Nessus"[10] simulated attack tool to attack a Linux server, and classified collected syslogs into three risk categories to create a corpus. We then used this corpus to

check logs (syslogs) for another attack by Nessus. As shown in Table 2, this method was able to classify logs with 95.5% accuracy.

Table 2 *Results of anomaly detection from logs*

Number of logs	Unknown/Safe	Warning	Danger	Total
	69	2	17	88
Correct	68	2	14	84
Incorrect	1	0	3	4
Accuracy	98.6	100	82.4	95.5

For comparison purposes, we conducted the same experiment on a Bayesian filter[11], which executes a similar statistics-based algorithm. A Bayesian filter is often used against so-called "spam" email. Since this filter can extract a small number of words from logs, we used trigrams instead of words. As a result, this filter was able to classify logs with 96.63% accuracy. The main reason for this increase in accuracy is that a trigram corresponds to several typical words (four characters or longer) within the corpus. Also, the filter selected the 15 most typical trigrams in the experiment; this helped eliminate intermediate values likely to cause noise.

In the category "Danger", we found three logs that should have been classified as "Unknown". All of these logs contained the string "rpc". The corpus also contained this string, whose value was used to indicate danger. These results show that when we attempt to detect anomalies based on words and few words are used to indicate danger, logs will likely be judged as anomalous based only on the presence of one of these words.

We compared the results between the Bayesian filter and our classification method based on the frequency of occurrence of words. Accordingly, we concluded that it is possible to improve the accuracy of our method by using the most typical trigrams in each category, as in the case of the Bayesian filter.

7 Conclusions

As described in this paper, we devised a system that can centralize log management through coordinated operation of log collection/management servers. We also developed the necessary coordination functions and a prototype of a log collection/management server that allows functions to be easily added or modified. Based on the results of functional verification of this prototype, we believe that this system will work effectively.

Essentially, this system consists of equivalent log collection/management servers, although performance will vary depending on the size of the organization or network in which the server is installed. However, when setting up a system across several organizations, it must be possible to vary the degree of coordination among the organizations according to their respective relationships. To this end, we are also studying a mechanism to vary the degree of coordination according to the communicating parties and whether communications are internal or external, through inter-organizational gateways that can restrict responses according to destination.

As we modify and improve this prototype, we will check the various individual functions and improve the performance to demonstrate that this system will work effectively in actual environments.

Acknowledgements

This research was carried out based on the "Research and Development on Security of Large-Scale Networks" project commissioned by the National Institute of Information and Communications Technology. We would like to express our gratitude to NICT for its guidance and support.

Reference

1 http://www.npa.go.jp/cyber/toukei/html/html18.htm

2 http://www.dmtf.org/standards/wbem

3 Distributed Management Task Force, Inc., "Common Information Model (CIM) Specification", http://www.dmtf.org/standards/cim/cim_spec_v22, Jun. 2004.

4 IBM Corp., "Tivoli Management Framework Planning for Deployment Guide Version 4.1.1"

5 Fukuda Naohiro., et al., "DAIKIBO NETTOWAKU SEKYURIT'I KAKUHO NI MUKETA KENKYU KAIHATSU", JNSA Network Security Forum 2003 (NSF2003), Oct. 2003.(in Japanese)

6 Ishida Tsunetake., Kamio Masakazu., Hakoda Takahisa., Katsuji Tsukamoto., and Hiroshi Shimizu., "Peer-to-Peer Routing Protocol for Reducing Network Traffic", 2004 IECE General Conference, No.B-16-20, p.629, Mar. 2004.

7 Masakazu Kamio., and Tsunetake Ishida.,"Unified Log Management and Abnormal Log Detection System", IPSJ SIG Tecnical Reports, Dec. 2004.

8 http://www.jnutella.org/docs/gnutellang/gnutella_protocolv4.shtml

9 L. Stoica, et al., "Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications", http://pdos.lcs.mit.edu/chord/

10 http://www.nessus.org

11 Graham, Paul., "Better Baysian Filtering", http://www.paulgraham.com/better.html, Jan. 2003.

KAMIO Masakazu

Corporation Product Division, Product Development Department, YASKAWA INFORMATION SYSTEMS Corporation

Computer Security, Multi-Agent System

ISHIDA Tsunetake

Corporation Product Division, Product Development Department, YASKAWA INFORMATION SYSTEMS Corporation

Computer Security, Computer Network

HAKODA Takahisa

Manager, Corporation Product Division, Product Development Department, YASKAWA INFORMATION SYSTEMS Corporation

Networks Security