ELSEVIER

Knowledge-Based Systems 14 (2001) 367–383

Knowledge-Based SYSTEMS

www.elsevier.com/locate/knosys

A multi-agent monitoring and diagnostic system for TCP/IP-based network and its coordination

Toshiharu Sugawara[a,*], Ken-ichiro Murakami[a], Shigeki Goto[b]

[a]NTT Network Innovation Laboratories, 3-9-11 Midori-cho, Musashino, Tokyo 180-8585, Japan
[b]Department of Information and Computer ScienceWaseda University, 3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169-8555, Japan

Received 20 September 2000; accepted 2 February 2001

Abstract

This paper describes an application of an AI-based multiagent system to the management and diagnosis of TCP/IP-based intranet/intra-AS (autonomous system) computer networks. A copy of this system is attached to each network segment and is made responsible for that segment. It captures packets in the promiscuous mode and analyzes their data in real time. Based on this analysis, the data needed to manage the local network are obtained, any changes in the local network or network components are recognized, and problems are detected. When a problem is reported by a user or detected by the system, the problem is diagnosed cooperatively or autonomously depending on its type. The activities of the agents are coordinated based on the concepts of coordination levels and functional organizations. An example of cooperative diagnosis clarifies why this multiagent approach is essential for network management. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Multi-agent application; Coordination; Internet; Diagnosis

1. Introduction

Computer networks (the Internet) are now indispensable in many fields. Ever-advancing network technologies have drastically changed not only how the computer is used but also the styles, methods, and environments of business, education, research, and daily life. Computer networks can lower costs and improve the timeliness and effectiveness of work-related matters. One problem with this heavy reliance on computer networks, however, is that any trouble in the network may result in lost business opportunities or crippled research projects. Networks must therefore be made to operate with little or no downtime, which means that network problems must be quickly resolved.

There are a number of AI-based management and diagnostic systems for one of telecommunication, wide-area, ISDN, ATM, and radio networks [1,9,11], but most of them focus on mainly layer-2 and lower-layer management. For constant network operations, the system that can monitor the network from higher-layer viewpoint is required. A number of SNMP-based (Simple Network Management Protocol) network management systems are commercially available. These systems can usually manage TCP/IP-based networks. However, these systems can only show network-related data to network managers; there is the still problem that the managers are required to analyze a huge amount of data, have a wide variety of computer and network knowledge, and run about the building(s) with the network analyzer. We believe that AI techniques are essential to overcome these problems. Furthermore, cooperative distributed management and diagnosis is a natural approach for geographically and functionally distributed networks [17], but many systems are still centralized; in these system, the scope of networks from the network management system is limited. Even if such systems were decentralized [16], there has been little discussion of how they might cooperate.

In this paper, we describe our application of a multiagent system called LODESv2 (*Large-internet Observation and Diagnostic Expert System*, version 2) for detecting and isolating problems of TCP/IP-based networks (with shared multiple-access media, such as Ethernet, FDDI, and token ring) in a cooperative manner. A copy of this system is attached to each network segment and is made responsible for that segment. The copy is called the LODES agent, which is an individually autonomous AI-based expert system with coordination ability. It captures packets in the promiscuous mode and analyzes their data in real time. Based on this analysis, the data needed to manage the local network are obtained, any changes in the local network

* Corresponding author.
E-mail addresses: sugawara@ntt-20.ecl.net (T. Sugawara), murakami@ntt-20.ecl.net (K.- Murakami), goto@goto.info.waseda.ac.jp (S. Goto).

0950-7051/01/$ - see front matter © 2001 Elsevier Science B.V. All rights reserved.
PII: S0950-7051(01)00150-2

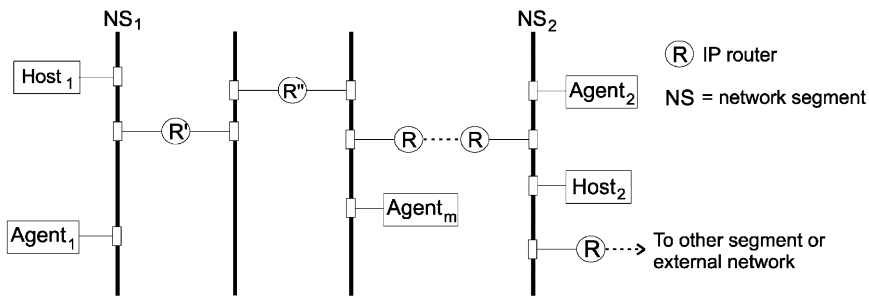

Fig. 1. An example of ISR problem.

or network components are recognized, and problems are detected. When a problem is reported by a user or detected by the system, the problem is diagnosed cooperatively or autonomously depending on its type. The previous version of this system (LODESv1, which was also a multiagent system) was described briefly in Ref. [12], but its basic design concept, especially from the viewpoint of a multiagent system approach, was not. Our new version, LODESv2, has three additional features that make it more efficient for real-world activities. First, coordination levels have been introduced for dynamically changing coordination strategies. They also separate descriptions of diagnosis from ones of coordination strategies. Second, each agent now has the ability to learn of coordination activities, to make collaboration more efficient. Finally, functional organizations have been added for specifying where an agent should send a message and with whom it should coordinate its activities.

In this paper, we discuss the significance, design concept, architectures, and cooperation mechanisms (including coordination levels and functional organizations) of LODESv2 from the viewpoint of multiagent approaches. The learning of coordination activities is discussed elsewhere [15]. This paper is organized as follows. First, we discuss why this kind of system is necessary and why it should be a multiagent system. Next, the system architecture and features of LODESv2, especially the local monitoring and coordinated activities, are described. Then, the expressions and inference mechanisms of its cooperative inference activities are presented. Next, an example of cooperative diagnosis for a problem that cannot be isolated by a single agent is described. Finally, its performance and further research issues are mentioned. LODESv2 is simply called LODES hereafter.

2. Motivation

2.1. Why a diagnostic system?

Computer networks (the Internet and TCP/IP-based network) now play an important role in many social activities. If a problem occurs in one of these networks, however, it can be quite difficult to troubleshoot for a number of reasons. First, these networks are functionally, physically, and geographically distributed. Because a number of nodes for network operations and services (such as DNS, DHCP, and Web servers) work together in a cooperative manner, network managers have to check these nodes, which may be located in different rooms and/or at distant sites. Moreover, a locally observed problem may actually originate in a node at a different site. Second, network managers must often analyze a huge amount of packet data because the number of packets during only a few seconds can easily be more than a thousand. Data analysis is thus not an easy task. Third, network problems can occur intermittently,[1] making them difficult to reproduce and troubleshoot. Fourth, the diverse growth of networks has led to different types of networks. A wide range of knowledge about computers and communications is thus required for troubleshooting. Finally, internetworking has dynamic features; for example, new hosts and new network segments can be connected, a host can be replaced by one running another OS, and routes can change over time. These dynamic changes make it difficult to isolate the cause of a problem. Furthermore, network problems can widely affect many people because a network is a shared resource. These situations are compounded in communities where users are not familiar with computer network technologies. There is thus a strong need for automatic network management and diagnostic systems that support easy, quick, and smart troubleshooting.

2.2. Why a multiagent system?

LODESv1 was initially intended to be developed as a centralized system. Within a few weeks, however, it became clear that it should be a cooperative distributed system for a number of reasons. The major reason was that there are problems incapable of being resolved or efficiently resolved without cooperation. For example, in the environment

[1] It is possible for a problem to look intermittent even if it is not. For example, suppose that several hosts have processes that send problematic packets (such as the ICMP netmask request broadcast shown in Fig. 4) periodically but not synchronously. Some time later (say 30 min to a week), these processes, from weak coupling, become inadvertently synchronized [3,10]. Each problematic packet results in a only small problem, but when they are sent simultaneously, they cause a serious problem. Because the synchronization is unexpected, this looks like an intermittent problem.

shown in Fig. 1, a user tries to send a telnet request from local host $host_1$ attached to network segment NS_1, but no response is received from target host $host_2$ in network segment NS_2 because the target host uses static routing, and it has no routing defined to $host_1$; thus, $host_2$ cannot reply to the telnet request (this problem is hereafter referred to as an *incorrect static routing* (ISR) problem). The network diagnostic system, $agent_1$, attached to the local network segment observes that the telnet request is actually sent to the appropriate router and that no reply is returned. However, it may not be able to diagnose this problem due to one of many possible causes: an intermediate router may have failed, the target host may be in the power-off state, the target host may have an incompatible netmask, the target host may have replied but the packet was lost somewhere, the reply packet may have an incorrect check sum, the TTL (time-to-live) number may be too small, and so on. On the other hand, no problem can be detected by the diagnostic system attached to the target network segment, $agent_2$, because a routing function is not used to communicate with hosts in the same segment; $host_2$ always responds correctly to test packets from $agent_2$, thus $agent_2$ believes that $host_2$ has no problem. Human managers resolve this type of problem by cooperating with the managers in charge of the remote network segments; a distributed multiagent approach is thus quite natural. How this example problem can be cooperatively diagnosed is discussed in Section 5.

Additionally, we must take into account how the Internet has been expanding, which is quite different from the expansion of other networks. For example, telephone networks were typically built by government agencies, initially in a top–down manner. A centralized system is thus a good approach to managing and diagnosing this type of network. In contrast, *inter*-nets are being built by connecting local, regional, and small-scale networks. A message transmitted to a destination may have to pass through other organizations and even other countries. This poses the question, should a centralized system be allowed to probe the networks of other organizations and investigate them? This situation can also occur within the intranet/intra-AS internet of a large company, organization, or university. Human experts have to hold high-level talks for the purpose of troubleshooting, questioning whether or not the message actually arrived and was relayed to the correct gateway/router, without knowing the details of the network structures and operating policies of other organizations. Such natural activities of human experts have inspired the use of the multiagent systems technique.

3. Multiagent system for network management and diagnosis

In this section, we introduce our system for managing computer networks and diagnosing TCP/IP-related problems in a distributed manner.

3.1. Design concept

3.1.1. Local management

A basic concept of LODES is the *local management*, meaning that a copy of this system (each autonomous system is referred to as an *agent* in the following sections) is attached to each network segment and is made responsible for local events. Centralized systems that monitor all network components use a number of protocols for network management, such as ICMP (internetwork control message protocol) and SNMP, to determine the state of the entire network. The packets obtaining management information used by these protocols must traverse a number of segments, thus causing a certain amount of traffic. The larger the network, the heavier this extra traffic load. Furthermore, network management information may be unable to be obtained because of a network problem. In the LODES concept, each agent obtains and analyzes only the information needed to manage the local network. This not only reduces the non-local traffic load but also enables the agent to fully understand the state of the local network as described below.

3.1.2. Passive management

Local management reduces the agent's work, so *passive management* becomes practical. By 'passive management' we mean that, in a realtime manner, a system acquires network management information and manages the network by analyzing the data in packets captured in the promiscuous mode without sending or receiving additional management packets. For example, the 'ping' (ICMP echo request) packets sent to hosts that are communicating normally with other hosts are not needed. Moreover, LODES can create a model of the local network configuration and features by observing packets. This model involves in local hosts, routers, and servers (such as DNS and DHCP/BOOTP servers) and network states. It is also used to recognize network activities such as the current activity state of each host (from IP and TCP packets), changes in network states (from traffic and other statistics), the detection of new and replaced hosts (from IP addresses and MAC numbers), partial routing information (from routing protocol packets), the services provided by local hosts (from the port numbers of TCP and UDP headers), and the IP addresses of DNSs and routers (from DNS and routing packets). SNMP, ICMP, and other test packets are used to get necessary complementary information for building the local network model[2] and to diagnose network problems.

3.1.3. Problem detection

Passive management also enables the system to detect problems (semi-)automatically. When the network has a problem, special kinds/forms of packets, different

[2] Data that are obtained using SNMP include routing tables, interface types and bandwidth in the local routers.

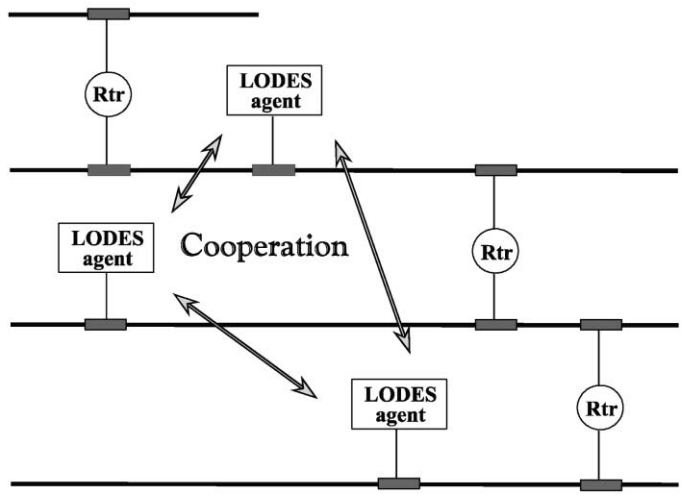

Fig. 2. LODES agents attached to segments.

packet-flow patterns, and/or abnormal statistical data are often observed. Their correspondence to possible problems can point to the cause of a problem. For example, a packet whose destination IP address indicates another network segment but whose target MAC number is the broadcast number or is not that of the router will be lost or duplicated. This phenomenon suggests an incorrect routing definition in the source host. As another example, multiple ARP (address resolution protocol) reply packets indicate multiple IP address assignments; this usually makes communication impossible. Not all problems can be detected in this manner. For example, a host that cannot send packets (because of a hardware failure or a software bug) will not be detected because there are no packets to observe. Such problems must be reported to LODES by network managers or host users.

This automatic problem detection can achieve effective packet capture. When a problem or an indication of a problem is observed, packets are stored in a timely manner for detailed analysis. This reduces the amount of data to be analyzed and thus reduces the inference time. Intermittent and potential problems can also be found automatically. For example, a 'broadcast storm' [3] may not exhibit any symptoms in a network segment if only a few hosts are attached, but symptoms may arise as more hosts are added. The agent can advise the network managers of this situation.

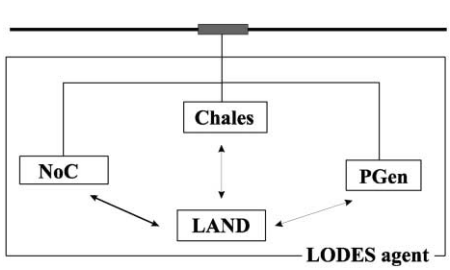

Fig. 3. Agent structure.

3.1.4. Cooperative monitoring and diagnosis

Local management limits the cognitive area of an agent. Cooperation among agents overcomes this limitation for non-local problem diagnosis. Although each agent has a detailed management model of its local network segment, some problems require coordinated activities such as requesting tasks and/or getting information about the problem symptoms and non-local network models.

In general, actions are coordinated if their results will be used by other agents and/or if doing so will help balance the load. Coordinated actions are determined based on each agent's world model, which is subjective but contains a partial global perspective. Thus, they usually include non-local data (such as the inference process and planning states of other agents and the domain-level data in other agents) as well as local data.

In LODES, coordination is used for more specific purposes.

Non-local problem detection. Symptoms of a problem may be observed in a network segment at a distance from the segment in which the problem originated. For example, the ISR problem (cf. Section 2 and Fig. 1) does not make local communications (between hosts in the local segment) impossible because local communications do not use a routing table. This problem can be observed only on other network segments.

Problem notification. It is often necessary for an agent to not only detect a problem but also notify the agent in the segment where it originated. This is not simple because the remote agent often cannot observe any problematic symptoms. It is thus necessary to send evidence of the problem to the remote agent so that the remote agent will change the hypothesis and plan it currently holds. Again in the example of the ISR problem, the routing function is never used in local communications; all tests in the local segment (i.e. those run by $agent_2$) show that the local host ($host_2$) is working correctly. Thus, $agent_1$ must send evidence of the problem to $agent_2$.

```
...........

ICMP 6:53:38 410ms x:x:x:0:a8:8 x:x:x:1:2:2a Host1 > hostX I:14950
         ttl:255 C:0x1aa0 TYPE(18) Netmask reply 255.255.0.0
UDP 6:53:38 410ms x:x:x:0:6f:8 x:x:x:0:d1:8 HostA.nfs > HostB.1023
         ttl:255 L:136 RPC(ID:-1444505557 T:1(Reply) S:0 VRF:0)
ICMP 6:53:38 410ms x:x:x:0:31:9 x:x:x:1:2:2a Host2 > hostX I:19403
         ttl:255 C:0x93a TYPE(18) Netmask reply 255.255.0.0
..................

ICMP 6:53:38 410ms x:x:x:0:2e:9 x:x:x:1:2:2a Host8 > hostX I:34054
         ttl:255 C:0xd001 TYPE(18) Netmask reply 255.255.0.0
ICMP 6:53:38 410ms x:x:x:0:2d:9 x:x:x:1:2:2a Host9 > hostX I:49597
         ttl:255 C:0x934b TYPE(18) Netmask reply 255.255.0.0
* ICMP 6:53:38 410ms x:x:x:0:ed:8 x:x:x:1:2:2a Host10 > hostX I:22119
         ttl:255 C:0x11d4 TYPE(18) Netmask reply 255.0.0.0
ICMP 6:53:38 420ms x:x:x:0:2c:9 x:x:x:1:2:2a Host11 > hostX I:38617
         ttl:255 C:0xbe30 TYPE(18) Netmask reply 255.255.0.0
..................

ICMP 6:53:38 420ms x:x:x:0:16:b x:x:x:1:2:2a Host19 > hostX I:15598
         ttl:255 C:0x2a58 TYPE(18) Netmask reply 255.255.0.0
UDP 6:53:38 420ms x:x:x:0:d1:8 x:x:x:0:6f:8 HostB.1023 > HostA.nfs
         ttl:255 L:120 RPC(ID:-1427728341 T:0(Call) V:2 PROG:100003(nfs))
..................
```

Fig. 4. Example of captured packets when a problem is detected. This is an example of a detected problematic situation in which all hosts in the local segment (more than 200) simultaneously replied to an ICMP netmask request [4] (this format for printed packet data is similar to that produced by Van Jacobson's tcpdump program for human network experts). This data is read and analyzed by the LAND component. Before these reply packets were observed, many ARP request packets (broadcast) and their replies were observed though they are not shown in this figure. Furthermore, the host Host10 had a different netmask (so this capture is done for two reasons, 'many ARP request flow' and 'incompatible netmask'). This is a typical intermittent problem that causes network congestion and soon disappears. However, the automatic detection reduces the amount of packet data that must be analyzed. This is quite important for both efficient system and human diagnosis. In conventional protocol analyzers, even if packets are captured, it is often difficult to identify data indicating a problem; to use the filter and search function, network managers have to predict what type of problem has occurred, and if this prediction is inappropriate, the problem is not located.

Network model comparison. The model of the local network and a partial (incomplete) model of the non-local network are exchanged by agents to help them to understand the configuration of the non-local network segments and/or to verify their generated models. For example, the ISR problem is observed only by $agent_1$, while $agent_2$ thinks that $host_2$ can communicate with any other host because it does not observe any malfunction. Even if $agent_1$ sends evidence that suggests $host_2$ is not working correctly, it is meaningless if $agent_2$ does not analyze the data appropriately. In such situations, a comparison of the network models may generate a sensible suggestion or an answer. In this example, the models of hosts differ. By identifying differing parts and using diagnostic knowledge about them, LODES can be associated with a number of specific hypotheses, one of which is the ISR problem.

Task and data requests. Tasks can be run to balance the load and to select and verify hypotheses. Requests for data needed to solve a problem may lead to new tasks in the receiver agents. In the ISR problem, for example, $agent_2$ may think that the route between itself and $agent_1$ is temporarily congested, making it impossible for the requesting service to receive reply packets. To verify this, $agent_2$ requests $agent_1$ to 'ping to $host_2$' and 'ping to other hosts in NS_2'. Note that the results of network observation often differ between agents, so agents may perform the same actions even though they may be redundant.

Problem reproduction. Reproducing a problem is one way to convince an agent that a problem exists. It can also enable an agent to capture timely data. Reproducing and observing a problem, however, often requires coordinated activities. Again in the ISR problem example $agent_2$ never observes the ISR problem so it does not recognize that its local host has a problem. In this case, $host_1$ (or $agent_1$) sends test packets, and $agent_2$ looks at them and at the packets sent in reply to them. This coordinated reproduction and observation will convince $agent_2$ of the existence of a problem in $host_2$. These actions can also be seen as synchronization, a type of coordination activity. Note that the help of human managers or users is sometimes necessary for reproduction.

In LODES, the agents are homogeneous even though they are situated in different environments. Their automatic management and troubleshooting capabilities are identical; only their local network data is different. Because they look at a problem differently, they generate different hypotheses about what is causing the problem and develop different plans about what to do next.

3.2. System configuration and architecture

The global configuration of LODES is shown in Fig. 2. A LODES agent is attached to the network segment for which it is made responsible. When a problem is detected or is reported by a network manager, a number of selected agents start the diagnostic processes. Diagnosis is performed cooperatively or autonomously depending on the problem type.

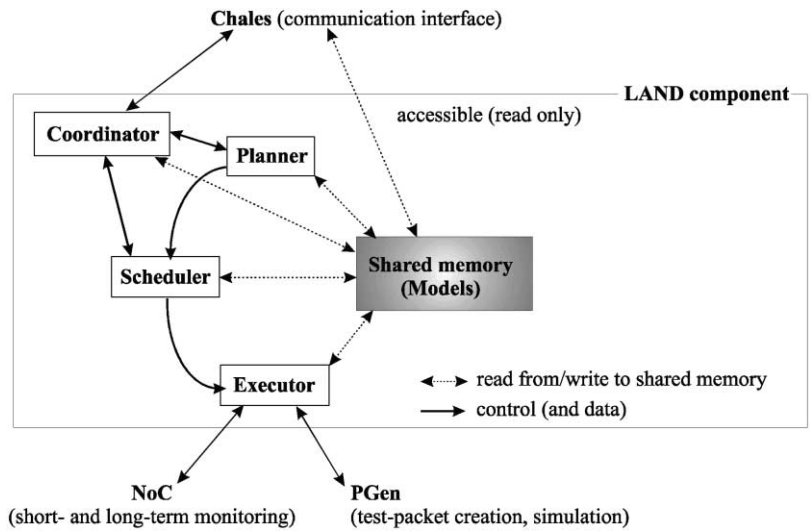

Fig. 5. LAND architecture.

The structure of a LODES agent is shown in Fig. 3. Each agent consists of four components: a *network observer component (NoC)* (this component corresponds to NOBS in LODESv1), a *network packet generator (PGen)*, a *local network diagnostic system* (LAND), and a *communication interface for large TCP/IP-network diagnosis expert systems (Chales)*. NoC captures packets in the promiscuous mode and analyzes them in real time, so it:

1. Generates management information and statistical data concerning its local network
2. Locates network problems based on its analysis of the captured packets and its embedded knowledge.

An example of packets captured when a problem is observed is shown in Fig. 4. To make this detection efficient, NoC uses only simple pattern matching. It may issue a warning even if there is no real problem, because LAND determines whether or not it is really a problem. The detection process thus errs on the safe side so that no serious problems are missed. The detection of a problem by NoC initiates a diagnostic process in LAND. PGen sends test and management packets (such as SNMP), receives their responses and/or simulates communication behaviors under the control of LAND. It also reports to LAND the amount and number of packets sent or received during each operation indicated by LAND. These data are stored in the LAND knowledgebase as history to achieve self-monitoring; they can be used to predict the resources being used and effects on the environment of performing an operation. LAND is the main management and diagnosis component. It uses SNMP and a number of test packets to gather management information that cannot be obtained simply by observation. It controls all other components during this diagnosis. Chales makes connections to other LODES agents, sends the messages requested by LAND and receives all messages from other agents.

LODES has knowledge involved in: (1) communication protocols (such as TCP/IP/Ethernet (IEEE 802.3) protocol suites[3] [4], DNS, dynamic routing protocols), (2) network management protocols (such as SNMP and ICMP), (3) problem detection, (4) diagnosis, and (5) cooperative work (how and when to communicate, and with whom). All necessary management data are generated by NoC and LAND (if all routers in the local segment can speak SNMP). These data include all types of addresses (including IP addresses, MAC numbers, IP broadcast address, frequently used multicast addresses, and the subnet mask), servers providing public services (including DNSs, Reverse ARP servers, DHCP (BOOTP) [7] servers, gateways, and IP routers), behavioral data on each host (services provided by the host and how to react to problematic packets), the types of lines to adjacent segments (such as the throughput and MTU), local physical and logical topologies, and IP addresses of other LODES agents. The details of NoC, PGen, and Chales are outside the scope of this paper since this paper focuses on multi-agent diagnosis; they are described elsewhere [12].

3.3. Structure of LAND component

The LAND component is in charge of diagnosing problems in cooperation with the other LODES agents. This diagnosis is based on the locally generated model about the world. The model includes partial non-local data (domain data in and inference states of other agents), as well as the locally obtained domain data (such as network configuration, management data, and observed current network states, as shown in Section 3.2) and the local inference states (e.g. the current hypothesis and plan and scheduled operations). The model is expressed by a set of variables stored in

[3] Currently, this system assumes that the datalink network is Ethernet or IEEE 802.3 because these are the most widely used ones.

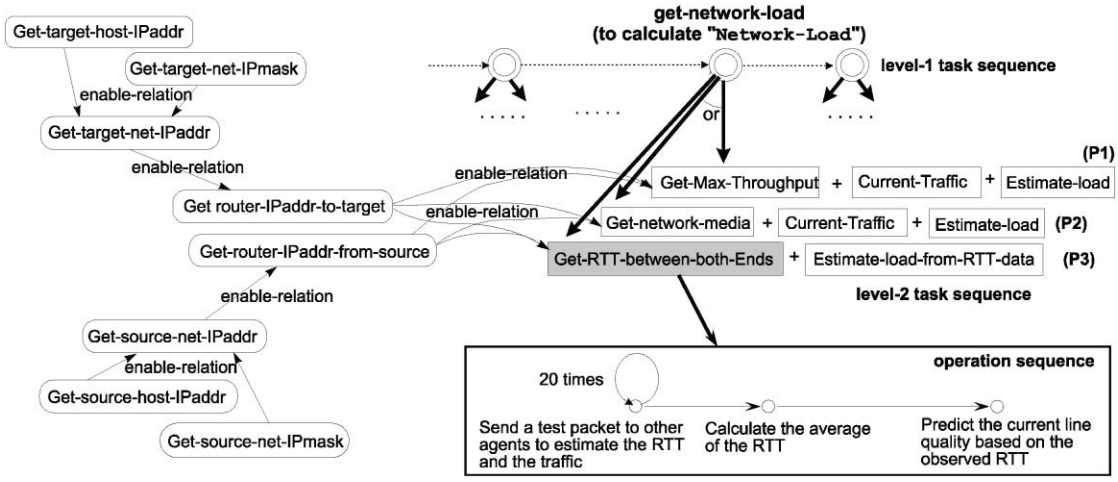

Fig. 6. Example of Generated Plan and Task Dependency Structure. Plans in this example are generated to investigate the local network traffic load, that is, to calculate the value of variable `network-load`, which is used in the current hypothesis verification. The task sequences in level-2 are generated to calculate this value. There are three sequences but one of them will be selected in the Planner. The selected sequence is further expanded to the operation sequence (only the first part of the sequence corresponding to the first element of the level-2 task sequence is shown in this paper). Of course, these operations also refer to other variables (enable-relation as shown in this figure), but they are scheduled to calculate in previous part of this sequence or have already been calculated in the previously selected plans.

the shared memory in the LAND component. The model created in LODES is explained more in Section 4.1.

The LAND component consists of the four subcomponents shown in Fig. 5. The Planner, based on the (locally) generated model, proposes possible causes of a problem (such a cause is called a *hypothesis*) and gives each one a likelihood rating, which is called the *certainty rating* or simply the *certainty* and is expressed by a positive number. The certainties of the hypotheses are initially given as diagnostic knowledge; but later they reflect past problem-solving experience (such as the number of occurrences) and the network data observed during the current diagnosis. The planner then builds a number of operation sequences to determine whether the candidate hypothesis is a valid explanation for the problem. After these operation sequences[4] for verifying the current hypothesis have been generated, the Planner selects the most appropriate one from its local viewpoint. This selected sequence is called the *current plan* or *cplan* for the current hypothesis in this paper. How to generate operation sequences is described in Section 4.

The scheduler then selects the operation that should be done next among the set of operations according to a partial order: if it is a local operation, it is executed by the executor; if the operation requires an operation execution or a variable value in other agents, the scheduler commits the coordinator to send a request message to other agents. All possible hypotheses, the selected hypothesis, the generated operation sequences, the selected plan, and the scheduled operations are stored in the local model as the inference state of this agent. The execution of operations defines/modifies the variable values in the local model (shared memory) as domain data.

The coordinator sends the committed messages and analyzes the received messages. If a received message requests an operation that has not yet been executed, this requested operation is also stored in the model. If the requested operation/variable is already executed/calculated its result/value is available from the shared memory, so its reply is immediately generated in the Coordinator. The Planner and the Scheduler look at the revised model after an operation has executed: the Planner decides whether the current hypothesis should be retained or another one should be selected; the scheduler decides which operation (a locally proposed one or requested one) should be executed next. This cycle is iterated until the candidate hypothesis is verified.

4. Inference and cooperative activities in LODES

4.1. Model and inference

4.1.1. TAEMS framework

It is assumed, for generality, that agents decide their actions according to TAEMS, a task-centered coordination model proposed by Decker and Lesser [5] based on generic coordination relationships. In this model, each agent makes planning and scheduling decisions based on a subjective view of its own and other agents' task structures, which

[4] More precisely, they are partially ordered sets of operations. The selected ordered set is serialized so that the operations are executed by the Scheduler using coordination knowledge. How to verify a hypothesis is not unique, so a number of ordered sets for verifying it are generated.

```
(defhypothesis
  IF (string= #V:Main-Symptom "No-Response-to-Service-Request")
     .........
     (string= (get-network-portion #V:*next-hop*) #V*Local-Network-Address*) #Z:3
     (eql #V:find-pingponged-ICMP-packet T)
  Then pingponged-packets
  Initial-rating 5
  Learned-weight 0)
```

This is the definition of the hypothesis `pingponged-packets` with initial rating 5 (rating is between 1 to 10), which describes the probability of the occurrence of this problem. Of course, the occurrence depends on the network environments; the rating is modified through experience. This definition expresses that the local router relays the packets for `NetIP-of-Rhost` to another local router so it is probable that packets to the `NetIP-of-Rhost` are pingponged between local routers. Note that the defhypothesis is invoked when all #V-signed variables are bound. All of the IF part is an AND condition; if an OR condition is used, the defhypothesis is copied into a number of `defhypothesis` definitions to eliminate the OR condition.

```
(defhypothesis
  If (string= #V:Main-Symptom "No-Response-to-Service-Request")
     .........
     (eq #V:Echo-Reply-from-the-Local-Host T)
     (string= #V:Sync-Ping-Observation "arrive-but-no-response")

  Then No-Routing-Data-in-Local-Host
  Supported-by (and (eq #V:Echo-Reply-from-the-Local-Host T)
                    (eq #V:(Echo-Reply-from-the-Remote-Host in Remote-Net agent) nil))
               #Z:5
  Initial-rating 2
  Learned-weight 1)
```

This definition of the hypothesis `No-Routing-Data-in-Local`-Host expresses that if the local host correctly responds to an ICMP echo request, no response is observed at a remote agent, and the local agent cannot observes the response packet to the ICMP echo request sent from the remote agent (there are some additional conditions, but they are omitted here), then this problem is `No-Routing-Data-in-Local-Host` (that is, the local host does not know about the routing relay to the remote network). Note that `Remote-Net` indicates the agent at the other end if this hypothesis is at the source- or target-side agent (See Section 4.5), otherwise this hypothesis is ignored.

Fig. 7. Examples of hypotheses.

consist of the relationships among these tasks such as *enable*-, *facilitate*-, and *support*-relations[5] and resource usage patterns (such as the *use*-relation) [6].

During diagnosis, the hypothesis having the highest certainty is selected. The certainty of the hypothesis is calculated from the rating initially given, the learned weight, and additional weights based on data observed during diagnosis and described by support-relations in TAEMS (the details are described later). Then the Planner has the highest-level (i.e. most-abstract-level) plan 'verify the selected hypothesis'. In this sense, a hypothesis can be identified with the corresponding highest-level plan, which is 'to verify it'. This plan is expanded into a number of task sequences based on the task structure, especially enable- and facilitate-relations, defined in TAEMS. Each task is a subplan or an operation. A subplan is further divided into a number of task sequences. An operation is a non-divisible executable form of a program or a shell script (Fig. 6). All

[5] The support-relation is a new relationship that was not in the original formulation discussed in Ref. [5]. This relates to how a task in one agent can affect the subjective view of its own and other agents' task structures by changing the importance rating of its tasks. When a task is requested of other agents that may have different goals, for example, the agent must decide which goal is more appropriate for the current problem. Even in a local agent, the problem situation is gradually understood by executing tasks; thus, the agent may have to decide that the current goal is not appropriate but another goal is. Support-relations, therefore, express the semantics of inference in the domain, although enable- and facilitate-relations are relatively syntactical. A support-relation may be qualitative or quantitative depending on the system designs. This rating change can, in turn, cause the agent to choose one task over another for execution.

```
(defplan "observe-pingponged-packets"
  :level 2    ;; describe abstract level (higher is more abstract)
  :used-variables IP-Address-of-RHost NetIP-of-Rhost *next-hop*
  :target-variables find-pingponged-ICMP-packets ;; this plan is invoked to bind
                                                 ;; this variable.
  :history-list-of-sequences  ;; sequence of operations generated by planner
  (((Start-capturing-ICMP-Echo-packets)
    (Send-Echo-Request-to-Rhost-and-Observe-Its-Reply)
    (Stop-the-capture)
    (Find-pingponged-ICMP-packets)
    :average-execution-time <execution-time> ;; Statistical data of execution
    ))                                       ;; time of this plan
  ................
  )
```

This subplan is invoked to bind the variable `find-pingponged-ICMP-packets`, which is partly used to verify the hypothesis `pingponged-packets` from the variables `IP-Address-of-RHost` `NetIP-of-Rhost` and `*next-hop*`. This plan may be executed when all variables in the assumed-variables slot are bound. If some variables in this slot are not bound, a (sub)sequence of operations for obtaining these values will be added into the plan to verify this hypothesis. When this subplan is completely done, the variable `pingponged-packets` is bound (and if the value of `pingponged-packets` is `T`, the hypothesis is verified).

Fig. 8. Example of a subplan.

subplans may be expanded into operations a priori, or some of them may be retained as subplans and expanded in an opportunistic manner (an example is shown in Fig. 6).

4.1.2. Model description

The (subjective) world model in a LODES agent is expressed by a set of variables and their values. This model includes the inference states of local and remote agents, including their current hypotheses, plans (operation sequences) and scheduled operations, as well as their domain-level data describing the network environment, the status of observable network elements, and the results of testing and operations. The part of the model describing remote agents and remote networks may be incomplete and insufficient.

We assume that because an agent selects a candidate hypothesis and generates a plan to verify it based on its subjective model, the values of the variables that are elements of the model do not frequently change. This assumption is necessary to guarantee that the inference activities based on the current hypothesis and plan remain feasible for some time. In a networking environment, data can change frequently. For example, the state of a network interface may frequently change (up and down). In this situation, the variable in the world model expressing the network interface is said to be 'flapping'. In another example, an agent may be observing a network state, such as network traffic, for some time. The number of bytes flowing in a network segment can greatly vary, but the variable used to describe the state in the model simply takes the value '(severely, slightly) heavy, normal, light, or inactive'. Frequently changeable variables are used in operation execution but not in hypothesis selection or plan generation.

4.1.3. Descriptions of hypothesis plan and operation

A candidate hypothesis is selected by using initial ratings of hypotheses and the agent's past experience. Two examples of hypothesis definition are shown in Fig. 7, where a #V-signed variable is one in the model of an agent and must be bound for verifying the hypothesis; when the IF part of the rule is satisfied, the hypothesis is verified. However, because some variables in the IF part are usually unbound at some point during the inference process, the plan is generated to obtain these unbound variables.

A (sub)plan and operations are defined by `defplan` (Fig. 8) and `defoperation` (Fig. 9), respectively. In Fig. 8, the variables in the `used-variables` slot are the ones that must be bound when this plan is executed (because they are referred to). The variables in the `target-variables` slot are the ones whose values this plan aims to calculate. The level slot is used to express the abstract level of task sequence for efficient planning (described later). Fig. 9 defines the procedure, which is an executable (LISP, perl) program or a shell script, invoked in this operation. The #D- and #V-signed variables in each operation describe variables whose values will be set or will be referred to during execution. The enable-relations of the task structure in TAEMS are generated from these dependencies in `defhypothesis`, `defplan`, and `defoperation`.

Because the agents are placed in different segments of the network and the results of tests and observations will vary with the observation point, it is necessary to describe which

This is an example of the operation "Router-Relay-to-Remote." #V:<variable> means the assumed variable that must be bound before this operation is invoked. #D:<variable> indicates that this variable will be bound after execution of this operation. These variable relations (dependencies) are used to generate a plan (a sequence of operations).

```
(defoperation Router-Relay-to-Remote
  :procedure
  (progn
    (if (null (setq #D:*next-hop*
                    (get-router-to-remote-by-snmp #V:NetIP-of-Rhost)))
        (setq #D:*next-hop* (get-router-to-remote-by-snmp "0.0.0.0")))))
```

The variable NetIP-of-Rhost, which is a string of the network IP address (not host address) of the remote host Host2 expressed by dotted notation, must be bound before this operation is invoked. This operation will bind to the variable *next-hop*, the IP address of the router relay to the remote network.

The function get-router-to-remote-by-SNMP sends an SNMP message to the local router (R' from the viewpoint of agent1 in Figure 1), which relays packets to the NetIP-of-Rhost; the function then gets the string of the IP address of the next router (R'' from the viewpoint of agent1 in Figure 1). This value is obtained using

```
(snmpget <router> "ip.ipRoutingTable.ipRouteEntry.ipRouteNextHop.<NetIP-of-Rhost>")
```

The 0.0.0.0 means the default routing. This function returns NIL when the local router does not look at SNMP or has no routing information for remote network.

Fig. 9. Example of an operation.

agent calculates the variable value. This type of variable is described in the form of (⟨variable⟩ ⟨agent⟩), such as Echo-Reply-from-the-Remote-Host in Fig. 7.[6] On the other hand, there are a number of variable values that do not depend on the observation point, so all agents get identical values for these variables. For example, the value of a variable computed using SNMP data from a specific router/gateway/switch/hub/host is invariant among agents. This type of variable is marked *invariant*.

4.1.4. Hypothesis selection and plan generation

In the beginning of diagnosis, the first candidate hypothesis is selected as follows. First, the hypotheses whose IF parts are consistent with the current subjective world model are collected. There are a number of variables that are always bound at the start of diagnosis, and these variables are used for efficient hypothesis selection. For example, the variable main-symptom describes the main symptom (such as no-answer-from-a-host or many-broadcasts-observed) of the problem reported by a LODES user or by an initial message from NoC. All hypotheses defined by defhypothesis are encoded into the tree structured by these variables. Then, from among these consistent hypotheses, one is selected based on its certainty; that is on the combination of its predefined rating (initial-rating) and experience (learned-weight).

Then the plan is generated to verify the selected hypothesis. To verify it, all variables in the IF part of the hypothesis definition must be calculated and the conditions must be satisfied. The Planner selects the referred variables that are still unbound and generates plans by tracing backward along dependency chains (enable-relations) from each of the unbound variables. That is, it finds the plans or operations in which (at least) one of these variables is set. The plans and operations that are found in this step usually require other variables that may be unbound, so other plans or operations are searched for in order to calculate one of these. This cycle is iterated until all unbound variables are included. This plan search is done from higher-level plans (defined in the level slot) to operations. Thus, a number of elements in the generated sequence (partially ordered set) are (sub)plans; these plans are further expanded into operation sequences by repeating the backward tracing, but some of them are done in another agent because the referred variables can only be calculated there. Note that the generated sequences for verifying a certain hypothesis are not always identical, because which variables are unbound (i.e. which operations were executed before) are different.

[6] In this example, the ⟨agent⟩ part is described using functional organization, which is explained in Section 4.5. The variables whose ⟨agent⟩ part is not specified have locally calculated values.

This is a part of the definition of structure `operation` corresponding to an operation defined by defoperation. A number of slot values (such as `No-of-pkts`, `real-time-cost-list`, `.value`, and `why-plan-propose`) are dynamically defined during or after the execution of the operation, although others (such as `referred-variables`, `defined-variables`, and `name` are bound when this operation is defined.

```
(defstruct operation
  name          ;; name of this operation
  default       ;; nil or list of default values
  type          ;; nil, costly, or side-effect
  called-for    ;; this operation is invoked to verify this hypothesis
  iteration     ;; If this is T, this operation can be executed two or more time.
  procedure     ;; program (Lambda formula, LISP source code) to be executed
                ;; when this op is invoked.
  ft            ;; program name. (after compilation, this function name is
                ;; used instead of invoking the above lambda formula or S-expressions
  args          ;; supplied arguments
  applied       ;; T if this operation is executed.
  nth           ;; Integer: this is the Nth operation during the last diagnosis
 No-of-pkts     ;; the number of packets sent during execution of this operation.
  real-time-cost-list ;; Elapsed-time history for executing this operation
  cpu-time-cost-list  ;; CPU-time history for executing this operation
  total-cost    ;; Estimated cost as the combination of above two slots.
  init-cost     ;; initially given cost
  document      ;; comment about this operation.
  why-plan-propose     ;; static (defined in the rule file)
  referred-variables   ;; #V-signed variables that must be bound
                       ;; when this operation is executed.
  defined-variables    ;; #D-signed variables that will be bound by this operation.
  .value     ;; value returned by the lisp program (stored in the ft or procedure).
  .......
  )
```

Fig. 10. Part of the operation structure.

There are usually a number of operation sequences to verify a hypothesis; the agent selects one of them based on its probability of execution and some measures (such as expected CPU and elapsed time and the quality of the obtained result) and the inference history. An operation is transformed into the structure as shown in Fig. 10. The name of the operation, the procedure (LISP program, shell script, etc.), and its return value are stored there. Furthermore, history data about the operation such as whether or not an operation was completely executed, CPU time, elapsed time and how many packets were sent/received are recorded there after its execution. This information is used to select the cplan.

The certainty of a hypothesis may change after the execution of each operation, because a number of operations in the plan can result in unexpected values that are incompatible with the current hypothesis or that suggest another hypothesis. In particular, some operation results, that is, some variable values, often strongly support a hypothesis. To represent this phenomenon, the hypothesis definition may have weighted conditions described in its IF part with #Z-signed weight integers ranging 1 (default) to 5 (cf. Fig. 7). A weighted condition means that if it is satisfied, the certainty of its hypothesis increases. Because the default weight of each condition is 1, the certainty of the current hypothesis increases if one of conditions in its IF part is proved to be satisfied.[7] The certainty of a hypothesis also takes into account the conditions in the `supported-by` slot. The conditions express that they are not necessarily used to verify it, but the facts strongly support it. Note that the certainty of not only the current hypothesis but also all consistent hypotheses may change. After each operation execution, the planner investigates the certainties of the

[7] Thus the hypothesis that has more conditions in its IF part may have higher certainty. This can be explained as follows: if a more specific hypothesis is supported by many condition predicates, it is natural that the specific hypothesis is selected. If this phenomenon causes a negative effect, the use of `learned-weight` plays an important role in overcoming it. The hypothesis that does not have a very specific IF part but frequently occurs in a network segment will have a high `learned-weight` value by experience. How to calculate this weight is system-dependent and is not discussed here.

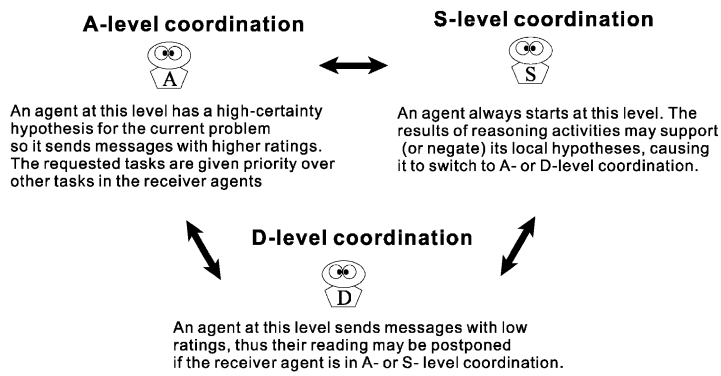

Fig. 11. Coordination levels in LODES.

current and other consistent hypotheses. If another one has higher certainty, the plan execution for the current hypothesis is suspended and the higher-certainty one is selected as the current hypothesis. It is obvious that the certainties of the hypotheses may also be affected by messages from other agents.

4.2. Types of coordination messages

Communications take place to achieve coordinated activities described in Section 3.1. Three types of messages are used in LODES.

- Request-type messages:
  Request for domain data, inference-state data (the current hypothesis, generated plan, executed operations, etc.): The variable names, which are elements of the agent's model, are included in this message. A message requesting domain data may induce a number of associated plans and operations in the receiver agents to investigate them.
  Request for operation executions: The names of the operations are included in the message.[8] This type of message is usually used for coordinating activities (see Section 5).

- Inform-type messages:
  Transmission of domain data and inference-state data: A pair of the variable and the value corresponding to the data item is transmitted.

- Control-type messages:
  Control (for synchronizing activities and stopping/interrupting the inference process): A control instruction name and its arguments are sent in this type of message.

A message has an associated rating indicating the importance of its contents as determined from the sender's viewpoint. The rating of the message is the certainty of the current hypothesis because the message is generated according to the plan (operation sequence) for verifying it.

4.3. Coordination levels

Three coordination levels, — nearly autonomous coordination (A-level coordination), shallow coordination (S-level coordination), and deep coordination (D-level coordination) — are introduced in LODES for effective coordination of activities (Fig. 11). A-level coordination implies that each agent's local autonomous control is almost exclusively decided based on locally available data. In this mode, it is assumed that knowledge about the specifics of operations and their results in other agents will minimally affect the decision about what operation to perform next. Furthermore, analysis of messages received from other agents will unnecessarily slow down local problem-solving activities.

In contrast, S-level coordination involves requesting domain data and operation executions of other agents. However, the use of non-local knowledge to make control decisions is severely limited because information about the range of activities of other agents and the detailed features of their environments is not used. Control of local problem solving is determined using only a small number of parameters and ratings attached to request messages by sender agents [13] based on their local viewpoints.

D-level coordination implies that local control decisions involve the use of a partial global view of the hypotheses and plans in other agents [8]. This view is generated in agents by exchanging both non-local domain and inference-state data, such as their current hypotheses and selected plans, as well as the intermediary results and observed data of other agents Fig. 11. Agents may also use negotiation messages to resolve detected conflicts at this coordination level. This more global view enables each agent to better understand the problem-solving state of other agents, so control decisions about which local task to execute next are made to optimize the network rather than the local problem-solving

[8] A request for domain data can induce operation execution but cannot specify the name of an operation. If the value is already known, the operation(s) to compute it is (are) not executed. In the example in Section 5, the LODES agent sends a request for an operation in which ICMP echo-request packets are sent to the specified target.

performance. D-level coordination is also necessary to avoid the serious side-effects of interactions among agents and environments.

The basic concept of dynamically changeable coordination levels is that full coordination is not always worthwhile because it can lead to additional overhead for analyzing message exchanges and for formulating coordinated activities. For more efficient cooperation, a nearly autonomous or shallow coordination regime is often better.

(A) When an agent has a hypothesis whose certainty is higher than the given threshold and has a plan to verify it in the autonomous regime, additional messages and inferences for collaboration may cause needless overhead (A-level coordination). This situation can be described in TAEMS as the agent having the high-certainty hypothesis and its plan being completely expanded to the operation sequence (even though some operations must be done in other agents). Thus the agent believes that its central and/or autonomous regime is more appropriate.

(S) When a number of agents have hypotheses that have a not-low and nearly equal certainty, these are equally worth being verified. Uncoordinated behavior might cause incoherent and/or redundant behaviors. There is, however, a trade-off between selecting one of these hypotheses by collaboration (D-level coordination) and trying to verify all of them without collective behavior (S-level coordination). This trade-off involves other factors such as the amount of resources. Therefore, in such a situation, D-level coordination activities are sometimes required [14].

(D) Deep coordination is definitely required when:

1. All agents have only hypotheses whose certainties are low from the local viewpoint (some of these may have higher certainties from a more global viewpoint);
2. An agent is about to execute a costly (time-consuming, traffic-generating, such as long-term observation, monitoring, polling, etc.) operation because it may not be worthwhile to execute this operation (there might be other operations that could be executed before this costly operation); or
3. In general, an agent is about to execute an operation that will consume/use scarce resources (described by the use-relation in TAEMS) or must be done within a strict deadline. In these cases, the operations of different agents must be done in turn or selectively.

The agent chooses the coordination level at which to operate based on the certainties of its possible hypotheses. The LODES agent selects S-level coordination when it start diagnosis and in the case of (S). The assumption behind this non-full coordination is that there are sufficient bandwidth and computational resources available on the network to sustain a certain level of non-coherent behavior. However, eager coordination may result in inefficient diagnosis. An agent will switch to A-level coordination when it obtains strong proof about the cause of the current diagnosis (that is, the current hypothesis has a certainty higher than the given threshold) and is confident that the hypothesis can be verified without the suggestions from other agents (its plan must thus already be generated). An agent will switch to D-level coordination if it has only hypotheses with low certainties or which can only be verified by using costly operations (such as long-term monitoring, testing that generates many packets in narrow-band lines, such as VPN and leased line and/or side-effect operations, such as setting up monitoring in routers using SNMP). This is described in more detail in Section 4.4.

4.4. Coordination levels and control

In A- and S-level coordination, the plan/operation that should be done next is selected simply according to their ratings. If they are selected to verify the locally selected hypothesis, their ratings are calculated from its certainty: The ratings of the plan and operation that are induced from the request message reflect the rating of this request message. Furthermore, in these coordination levels, an agent sends only messages requesting domain data and operation executions in order to execute its local plan. Therefore, this non-full coordination does not need sophisticated decision for collaboration, but may bring about incoherent and redundant behaviors, so additional resources are consumed. However, in LODES, many (but not all) problems in this domain can be isolated using this request-based and simple rating-based collaboration, because they are not so complicated and the local network environment has sufficient network resources.[9] Note that if the agent is in A-level coordination the ratings of its messages are relatively higher, so the received requests are usually performed immediately.

In D-level coordination, an agent gets more non-local data by requesting inference-state data and coordinates its activities as follows. First, by analyzing the received not-low-rated plan of other agents, the agent identifies the parts it can execute instead of the agent that originally generated the plan. These parts (subsequences of operations) are usually the calculations of invariant variables. The agent then executes these parts of plans if the original agent agrees

[9] Non-full coordination may generate redundant activities and traffic (such as SNMP and ICMP packets for non-coherent testing and data collection), which is not acceptable in some network segments such as narrow-band VPN and leased-line. On the other hand, full coordination also generates additional packets for coordination. Anyway, it is better that the agent can predict how many packets will be sent/received through the scarce network resources during diagnosis. Fortunately, LODES has a monitoring component, NoC, that can count the input/output packets from/to LODES. This self-monitoring can be used to reflect its coordination strategy for future diagnosis. This learning ability is described in Ref. [15].

on this collaboration. Second, the agent evaluates hypotheses from a more global viewpoint, that is, using its local data, it re-evaluates the received hypotheses of other agents (this is possible because, although all agents in LODES have different domain data, they have identical diagnostic and control knowledge and the same inference mechanism). For this purpose, hypotheses are always sent with information about the functional organization (described in Section 4.5). For example, the `hypothesis No-Routing-Data-in-Local` in Fig. 7 may be strongly supported by the non-local variable `Echo-Reply-from-the-Remote-Host` (= NIL) of the agent in the remote-side network segment. Therefore, the agent in the remote-side segment has this data and (1) if its value is NIL, this hypothesis has higher certainty so it should be verified immediately or (2) if its value is not NIL, this hypothesis is not valid. These results are sent to the agent that formulated the hypothesis. Finally, to re-evaluate the hypotheses in other agents, an agent generates and executes a sequence of operations to identify the variable values that strongly affect the certainties of the hypotheses. Again in the same example, if the agent in the remote-side segment does not have the value of `Echo-Reply-from-the-Remote-Host` it generates a sequence of operations to obtain it. Moreover, if the sequence includes no costly operations, it is worthwhile executing it. Note that messages sent during D-level coordination have relatively low ratings. Thus, the request messages from agents in S- or A-level coordination take priority over ones from agents in D-level coordination.

The Coordinator component coordinates these activities. First, it manages message exchanges; it sends the messages committed by the executor and reads/analyzes the received messages. The analyzed message data is stored in the local model. Furthermore, by comparing the rating of the local hypothesis to the rating of the message, it determines whether the message is to be analyzed immediately or left for later analysis. Note that a reply to a request for inference-state data is generated immediately because the data are always available from shared memory. Second, the coordinator manages the coordination level of the local agent by monitoring the current and other possible hypotheses. Note that, we must consider the case where an agent has a high--certainty hypothesis (causing it to enter A-level coordination) but its plan contains a costly operation. This hypothesis is called a *marked hypothesis*. The time for executing the plan to verify the marked hypothesis is domain-dependent. The LODES agents hold the plans for the marked hypotheses until D-level coordination is selected; they are re-evaluated with non-local domain data and inference-state data, then executed. This is because it takes longer to execute such operations, but the marked hypotheses usually have higher certainties, so they usually have higher priority even in the D-level coordination regime.

4.5. Functional organization of agents

The concept of functional organization is introduced to indirectly specify with whom an agent should coordinate and to whom a message should be sent, because participating agents and their roles in diagnosis vary from problem to problem. In particular, the role of each agent depends on where the problem is observed, where the cause is located, and the structure and architecture of the network.[10] In LODESv1, the first few steps of diagnosis were always to used to identify the agents to coordinate with. This means that knowledge for diagnosis and knowledge for organizational role were not clearly separated so diagnostic knowledge easily became complicated.

In our problem domain, two types of organizations are mainly used (see Fig. 12) depending on whether the current hypothesis concerns a link-level (IP layer) or higher-layer (TCP/UDP- and application-layer-) problem. The cause of a link-level problem may be located in the source, target, or intermediate network. Furthermore, an IP-related problem may be caused by DNSs. The agents managing these components are required to coordinate based on their observational positions. The cause of higher-layer problems is certainly in one or both of the terminal hosts, so only the LODES agents managing these hosts need to be and should be coordinated.

In the description of agent's knowledge, other coordinating agents are denoted by the functional organization instead of by agent names or IDs. For example, in Fig. 7, the definition of hypothesis `No-routing-data-in-local-Host`, which is an IP-related problem, is strongly supported by the data indicated by the variable `Echo-Reply-from-the-Remote-Host` in the agent managing the other end of the network segment which is denoted 'Remote-Net agent' in this figure. Note that when an agent sends its hypothesis to other agents during inference, information on the functional organization (role) of the destination agent from the viewpoint of the sender is also attached.

In diagnostic knowledge, agents to coordinate with are always expressed by their roles in the functional organization. The role assignment process is necessary for actual coordination and communication. This process is invoked on demand, that is, when a message is about to be sent or just arrives. For this process, an operation sequence for assigning/verifying roles is generated using organizational knowledge and has priority over the normal plan.

5. Example of cooperative diagnosis and performance

An example of cooperative diagnosis is described in this section. The problem, which was briefly explained in Section

[10] The logical structure of a network is, for example, reflected by its routing. As another example, a problem occurs when the primary DNS is out-of-operation and the secondary DNS operates instead of it. So, in general, the roles of agents should be decided dynamically.

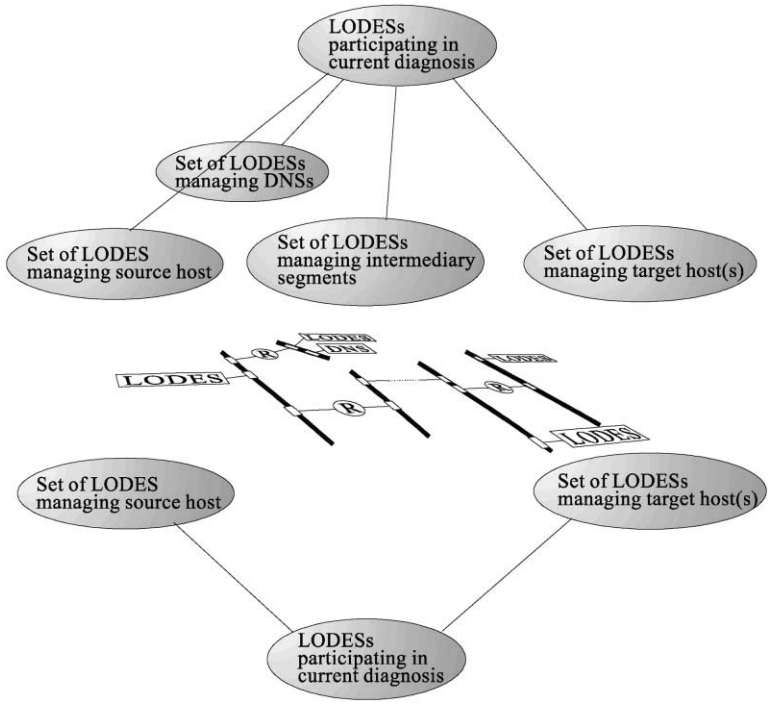

Fig. 12. Functional organization in LODES.

2, is caused by incomplete routing definitions in a host that uses a static routing table. The symptom of this example problem is that although a user, from local host $host_1$, has tried to establish a telnet connection with a remote host $host_2$, it cannot be established (normally the message from the telnet program is 'time-out') in the environment shown in Fig. 1. After this problem is reported by the user, the local LODES agent, $agent_1$, starts the diagnosis. It sends a diagnosis request to remote agent $agent_2$ because this may not be a local problem.[11]

$Agent_1$ see no indication of the problem except that $host_2$'s response to a test packet cannot be found; thus, it thinks that the problem's cause is not in the local network. All the consistent hypotheses in $agent_1$ have a low certainty. Note that an agent can have a number of hypotheses about the causes of problems in non-local networks. These hypotheses, however, have low certainties because the remote agents that manage the problematic network segments can verify these at lower cost. $Agent_2$ also sees no indication of the reported problem; that is, $host_2$ correctly responds to any kind of test packet sent by $agent_2$. Therefore, $agent_2$ creates a number of hypotheses; two in particular have higher certainties: (H1) it might be a TCP- or application-level problem or (H2) heavy (burst) traffic might be temporarily preventing $host_1$ from communicating with $host_2$. For H2, $agent_2$ investigates the history of messages from NoC that convey information about congestion. However, such messages cannot be found. To verify H1[12] $agent_2$, sends a telnet request to $host_2$ instead of $host_1$ but observes a correct response from $host_2$. Therefore, $agent_2$ thinks that this TCP- or application-problem occurs only when $host_1$ and $host_2$ begin to communicate. However, the plan for verifying this is costly because it requires that the user of $host_1$ reproduces the problem for packet observation. So, this plan is generated, but its execution is postponed. $Agent_1$ also generates a number of hypotheses, such as (H3) heavy traffic in an intermediate network. Unlike $agent_2$, $agent_1$ does not raise the certainty of H3 because it still cannot observe $host_2$'s response to test packets, but it can communicate normally with $agent_2$. Then because both agents have only low-certainty hypotheses, they enter D-level coordination.

By exchanging the possible but low-certainty hypotheses, $agent_1$ does not raise the certainty of H1 in $agent_2$ because $agent_1$ still cannot observe any response to ping packets from $host_2$. Then, their hypotheses related to IP-level (or link-level) problems are given priority. Furthermore, because $agent_1$ discovers that the condition supporting `No-Routing-Data-in-Local-Host` (see the definition of `No-Routing-Data-in-Local-Host`) is held, the certainty of the hypothesis increased. $Agent_1$ sends this result (and an explanation for it) to the

[11] NoC cannot detect this problem because it detects problems by observation, and no problematic packet from $host_2$ is observed. Strictly speaking, it is logically possible to detect this type of problem, but the process for detecting it is inefficient because a timeout process, which does not suit a realtime process, is required. Furthermore, this problem does not seriously affect other communications, so it is reluctantly give up trying to detect it.

[12] H1 actually consists of a number of hypotheses, each of which corresponds to a TCP- or application-level problem. These hypotheses are verified in turn.

originator of the hypothesis, that is, $agent_2$, who selects it next to verify.

$Agent_2$'s plan for verifying it (more specifically, the operations to compute the value of `Sync-Ping-Observation`) contains collaborative synchronized work with $agent_1$; $agent_1$ sends $host_2$ a number of ICMP echo-request packets; $agent_2$ observes them and $host_2$'s response to them. During this operation, $agent_2$ observes that the request packets from $agent_1$ arrive, but $host_2$ never replies to them, thus the value of `Sync-Ping-Observation` is '`arrive-but-no-response`'. This result may raise the certainties of other hypotheses, such as (H4) a $host_2$ interface failure and (H5) $host_2$ is running in single-user mode. However, these are easily rejected by simple testing. $Agent_2$ then determines that only `No-Routing-Data-in-Local`, which means that $host_2$'s definition for routing to the remote network is incorrect or missing, is a high-certainty hypothesis and the verified one.

This system runs on FreeBSD/Linux/SUNOS. LAND is implemented in KCL (Kyoto Common Lisp) or GCL (GNU Common Lisp) and C. Other components are fully implemented in C. NoC can capture and analyze more than 2000 packets per second, but this number varies depending on the traffic pattern and types of packets. The bottleneck in NoC performance seems to be not in the CPU but is likely to be in performance of its network interface and its driver. The LAND component can diagnose the ISR problem described above in around 60 s (elapsed time, not CPU time). This type of problem does not consume CPU resources but is time-consuming because the agent has to wait for the responses to test packets, observe the packets, and communicate with local gateways/routers and collaborating agents.[13]

This system currently works over Ethernet and fast Ethernet. The performance of NoC may, however, be insufficient for broader-band datalink networks, such as Gigabit Ethernet, FDDI (fiber distribution data interface) and ATM (asynchronous transfer mode). Furthermore, this system is implemented on a non-dedicated UNIX (FreeBSD) machine. NoC may, therefore, be swapped out or placed in the sleep state, causing it to sometimes drop a few packets. However, even in a faster datalink network, its performance is practical because the full use of a network's bandwidth is infrequent. This system also assumes that IP packets are not fragmented into smaller non-IP frames/cells. NoC would have to re-assemble fragmented frames/cells to check the IP and TCP/UDP headers in this type of network, leading to poor performance. We also have to consider networks using switching hubs. Fortunately, a number of switching hubs have a function to set up a watch-dog-mode interface via which all packets can be monitored. This may not enable the LODES agent to capture all packets, but if an agent can monitor packets to/from routers, it can diagnose link-level non-local problems.

[13] Especially in this example, $agent_1$ has to wait for $agent_2$'s responses to the ICMP echo-request packets, although they are never observed.

6. Consideration and discussion

A distributed configuration has a number of desirable features for network management and diagnosis (see also Section 1 of Ref. [2]).

Reliability. A fault in the management system does not mean a failure in managing the entire network; other systems can partially compensate for the losses caused by the fault.

Fault tolerance. Even if the network segments are separated due to a network problem, each connected network has an operating network management system.

High performance. Network data can be concurrently captured and analyzed. Additionally, management data can be stored in and maintained with distributed local management systems. These features increase the efficiency of network management.

Lower communication cost. Distributed systems do not send raw data but communicate by using high-level messages for coordinating activities. This drastically reduces the amount of communication. In a non-distributed system, all packet data must be sent to a centralized system node for analysis and diagnosis. This increases the network traffic load and may worsen the network condition.

Secure network management (Security). Distributed systems send only high-level messages. This feature is strongly required in institutions or agencies that deal with secret or confidential information, e.g. financial and military data. It may also be required within a company. For example, source programs and digital dictionaries are often allowed to be used by a restricted group of users, and their data should be kept from other people.

All of these features are quite important and indicate that cooperative distributed systems are suitable for network applications.

There are a number of issues concerning the topics in this paper. For example, agents should sometimes take into account multiple aspects of a functional organization. For example, a host connected to a third party's network segment can unexpectedly cause a problem; thus the agents, in parallel, have to form and maintain another functional organization involving a new agent to manage the third party's segment. In the current version, this type of problem is recognized as two independent problems. Similarly, a problem may cause symptoms at different points, so the agents may start and finish their diagnoses independently. It is necessary to generate a more global domain model for relating these independently observed symptoms.

Another related issue is to introduce the concept of proxy-like services. For example, a dedicated SSL node is placed between a client and a server and takes on SSL encoding and decoding instead of the server for load sharing. In the

current LODES, communication between the client and server like this is recognized as two dependent connections, namely the connection between the client and the SSL-node and the connection between the SSL-node and the server. However, it is better if these connections are recognized collectively as the involved communication. For this issue, we plan to add new knowledge for more a flexible communication model for diagnosis.

7. Conclusions

We have shown that a multiagent approach to network management is quite appropriate when we consider such network features as the geographical and functional distribution of networks. This approach also improves the fault tolerance of network management. We presented a multiagent application system called LODESv2. A LODES agent is attached to each network segment and these agents diagnose problems in cooperation with each other, with each one being responsible for its own network segment. An agent can identify the configuration of the local network structure and obtain management data by analyzing the flow of packets. These configuration data differ between networks and are difficult to maintain manually because of changes over time, so this identification of configuration is effective and useful. An agent can detect problems based on analysis of this data and gain knowledge of computer-network problems. This ability enables timely packet captures and detections of intermittent and potential problems. Then with the detection of a problem or with a problem report from a user (network manager) the LODES agents diagnose the problem to isolate the problem's cause. The diagnostic activities of these agents are coordinated for non-local problems based on the concepts of coordination levels and functional organizations. An example of cooperative diagnosis showed the benefits of using cooperation in this type of network management system.

Acknowledgements

I would like to thank Professor Katsuharu Tsukamoto, Kogakuin University, Tokyo, and Professor Rikio Onai, The University of Electro-Communications, Tokyo, for providing support for this project when it was at NTT Laboratories.

References

[1] S. Albayrak, Intelligent Agents for Telecommunications Applications, IOS Press, 1998 ISBN90-5199-295-5.
[2] A.H. Bond, L. Gasser, Readings in Distributed Artificial Intelligence, Morgan Kaufmann Publishers, CA, 1988.
[3] L. Bosack, C. Hedrick, Problems in Large LANs, IEEE Network Magazine 2 (1) (1988) 49–56.
[4] D. Comer, Internetworking with TCP/IP, Principles, Protocols, and Architecture, Second ed, vol. I, Prentice-Hall, Englewood Cliffs, NJ, 1991.
[5] K. Decker, V.R. Lesser, Quantitative modeling of complex environments, International Journal of Intelligent Systems in Accounting, Finance and Management 2 (4) (1993) 215–234, special issue on Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior.
[6] K. Decker, J. Li, Coordinating mutually exclusive resources using GPGP, Journal of Autonomous Agents and Multi-Agent Systems 3 (2) (2000) 133–157.
[7] R. Droms, Dynamic Host Configuration Protocol, RFC (Request for Comment) (1993) 1531.
[8] E. Durfee, V.R. Lesser, Partial global planning: a coordination framework for distributed hypothesis formulation, IEEE Transcation on System, Man, and Cybernetics 21 (5) (1991) 1167–1183.
[9] E.C. Ericson, L.T. Ericson, D. Minoli, (Eds.), Expert Systems Applications in Integrated Network Management, Artech House, ISBN ~ 0-89006-378-8, 1989.
[10] S. Floyd, V. Jacobson, The synchronization of periodic routing messages, IEEE/ACM Transaction on Networking 2 (2) (1994) 122–136.
[11] J. Leibowits, D.S. Prerau, Worldwide Intelligent Systems — Approaches to Telecommunications and Network Management, IOS Press, 1995 ISBN90-5199-183-5.
[12] T. Sugawara, K. Murakami, A multiagent diagnostic system for internetwork problems, Proceedings of INET'92, Kobe, Japan (1992) 317–325.
[13] T. Sugawara, Using action benefits and plan certainties in multiagent problem solving, Proceedings of IEEE Conference on Artificial Intelligence Applications (1993) 407–413.
[14] T. Sugawara, V.R. Lesser, On-line learning of coordination plans, Proceedings of the 12th International AAAI Workshop on Distributed AI, 1993. The longer version of this paper is available as COINS Technical Report, 93–27, University of Massachusetts, 1993.
[15] T. Sugawara, V.R. Lesser, Learning to improve coordinated actions in cooperative distributed problem-solving environments, Machine Learning 30 (2–3) (1998) 129–153.
[16] H. Velthujisen, Distributed Artificial Intelligence for Runtime Feature-Interaction Resolution, IEEE Computer 26 (8) (1993) 48–55.
[17] R. Weihmayer, H. Velthujisen, Distributed AI and cooperative systems for telecommunications, in: J. Leibowits, D.S. Prerau (Eds.), Worldwide Intelligent Systems — Approaches to Telecommunications and Network Management, IOS Press, 1995, pp. 227–260 ISBN90-5199-183-5.