第 XXX 部

実ノードを用いた大規模な
インターネットシミュレーション
環境の構築

第 30 部
実ノードを用いた大規模なインターネット シミュレーション環境の構築

第 1 章 Deep Space One ワーキンググループ 2005 年度の活動

Deep Space One ワーキンググループでは、実ノードによる大規模ネットワーク実験環境の構築技術や利用方法の研究に取り組んでいる。汎用的な大規模なネットワーク実験環境である StarBED や GARIT、セキュリティ用途に特化した SIOS や VM Nebula を通してさまざまな議論を進めている。

本報告書では本年度の主な活動について述べる。第 2 章では StarBED で利用している実験支援システム SpringOS の設計、第 3 章では SpringOS での VMware 制御、第 4 章では複数の実験環境の連携による大規模かつ複雑な事案の再現についての提案および統合手法についての考察、第 5 章では複数の実験環境を実際に接続しての実験報告を行う。

第 2 章 Automatic Configuration and Execution of Internet Experiments On An Actual Node-based Testbed

Software simulators are widely used for validation and evaluation of new network technologies and services. Using software simulators is a good way to validate algorithms or observing micro-behavior of communication protocols. There are, however, problems with software simulators. Most software simulators require target systems to be described under their own modelling scheme, often using their own modelling language. These descriptions are usually different from what will actually be running as products. It is clear that these products should be validated someway. Time required to run software simulation become problematic also, as we try to simulate realistic target system under realistic environment where non-trivial aggregation of complex network services come into play.

We adopt an approach to prepare a configurable testbed using actual nodes. Experiment topologies are created on this testbed virtually without changing physical connections, because the cost of building such experiment environments is very large.

Since users of such testbed have to perform many steps to execute the desired experiments on such environment, we design the system that supports the users to execute their experiments. Using our system, all the user have to perform is preparing a experiment configuration file. Our system will execute experiments according to the configuration file.

This paper shows the design of our supporting system models, steps of experiment with our system and an example of user's scenario.

2.1 Introduction

As the Internet continues to grow, the Internet has become the most versatile and most widely spread communication infrastructure that covers the entire world, with numerous critical services running on it. Therefore, the importance of evaluating new algorithms and their implementations has been increasing.

Software simulator is the most popular approarch to evaluate them, and ns-2[272] is the well known example of such simulators. However, we often cannot use implementations of target technologies directly on the Internet. Since they often require simulator specific implementations which is different from what will actually be running on the Internet. The implementations for simulators may differ from implementations for the Internet. We should evaluate target technologies,

including the bugs in the product implementations and behavior that is not described in the specifications before introducing new technologies to the Internet. When we perform an experiment that requires detailed action for many nodes, it takes long time with software simulators. On the other hand, if we use actual node based testbeds, the experiment will be finished in the actual time that described in user's scenario.

NSE[79] is a software to adopt actual node to ns-2. NSE's approach is using both physical network and software simulators. Using software simulator in a part of the experiment environment is efficient to simulate the scale of the Internet. Unfortunately, we have very little knowledge about new technologies, especially about their behaviors and influences to another system when they are introduced to the Internet. So deciding the parts where to use physical network or software simulators is difficult. We could not estimate the influences by software simulators to the result of experiments. There is also a problem of difference between real time clocking and simulated time clocking. In many cases, simulation speed dictates the entire NSE based setups, so this approach is still not immune to time problem.

Evaluating implementations with actual nodes which are used on the Internet is required. We employ an experiment environment using actual nodes and programs to evaluate new algorithms and their implementations. Using an environment built from actual nodes and programs, we can obtain realistic results. This approach also enables performance test under realistic environment. We assume that all resources are located in the same site, and all these resource are physically accessible. So we can get all of informations about the experiment traffic since all switches that connect experiment nodes are located in the same site.

However, to build such environment, we need a variety of equipments and installations, which make such kind of projects quite difficult to achieve. To solve the difficultly of building experiment topologies, the network will be beforehand built based on a fixed hardware connections and we build customized topology using virtual networks. We also decided to make our environment as a multi-user system.

Netbed[304] offers these functions and it adopt ns-2 like user interface. ns-2 is an event discrete simulator and use time as a event's trigger, thus it is difficult to use an event as another event's trigger for change experiment scenario by the result of the experiment itself.

We designed a support system which builds the desired experiment topology and drives the experiment following the user defined steps, automatically. Moreover, this system makes the synchronization of experiment nodes. In following sections, we describe the design of our system that automates the experiment environment settings and preparations as well as driving the experiment.

2.2 Experimental Environments and mechanisms to drive experiments

We have chosen the network based on actual nodes as an experiment environment to be as close as the Internet environment. However, it needs very large costs to build network using actual nodes.

To reduce the costs for building an experiment environment, we adopt the approach to share a physical network with several users. In this section, we describe the experiment environment with our approach.

2.2.1 Experiment environment

The network will be beforehand built based on fixed hardware connections and there are many kind of actual nodes. We can set the desired topology for the experiments using VLAN[119]. And then the experiment was driven on the environment. Thereby we don't have to change the physical topology.

Since the environment is dividable and could be manipulated by several users, concurrent resource

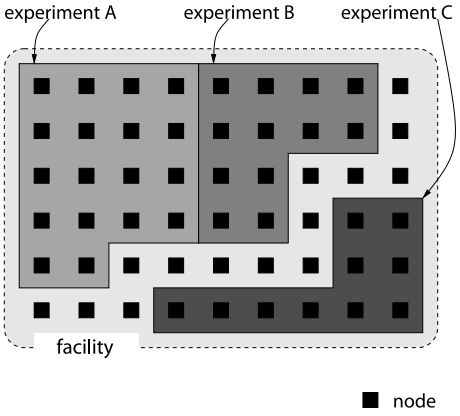

Fig. 2.1. Using facility by space division

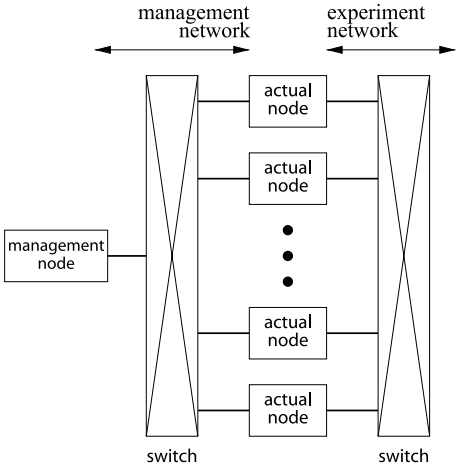

Fig. 2.2. Concept of experimental environment

usage can happen, such as nodes assignment or switch configurations. For that reason we need to have an access restriction and mediation mechanisms to share and manage those resources. Figure 2.1 shows a space division of resource allocation considering three independent experiments, each set of nodes is completely independent from the others.

The actual management mechanisms are deployed using a separate network, and each node the our environment has a network interface just dedicated for this purpose. By separating the management network from the network dedicated to the experiments, we guarantee the traffic separation and the precision of the experiments. Figure 2.2 shows this concept.

2.2.2 Execution of an experiment

In the previous paragraph, we presented the connectivity and the characteristics of our experimental environment as well as the needed preparations and management procedures that we assumed from here on. But during the experiment, we need to define various other procedures and that will be the topic of the following paragraph.

Nodes are assigned by the administrator to the users. Users setup these nodes by installing the needed softwares for their experiments. We call *pooled node* the actual node which is not used in experiments, we also call *leased node* the node assigned to an experiment and *experiment node* the node configured to be used in an experiment.

The Wake on LAN mechanism is used to boot up nodes. Experiment Nodes load OS using the bootloader provided by PXE[126]. The bootloader specifies whether the node starts as diskless system or boots from local partition. The bootloader configurations are specified by the DHCP[53] server. Before booting up leased nodes, the user may setup the PXE so that the node boots from a specified partition and using Wake on LAN to boot up nodes.

Virtual networks are identified by VLAN IDs, which will be used to build the network topology for an experiment. These virtual networks are created using configurable switches according to the needs of each experiment.

After building the network topology, the user needs to run their commands on each node to perform the experiment. We will call *scenario* a list of commands or execution steps of an experience.

And the resulting logs of an experiment are collected and analyzed by the end of the whole experiment process.

These procedures of generic experiment process are listed below:

1. A set of nodes and VLAN IDs are assigned to the user by the environment administrator.
2. The user setups required software into the leased nodes.

3. The network of the experiment is built by means of VLAN.
4. The user command list making the scenario, is executed.
5. Execution log files are collected from the experiment nodes.
6. Log files analysis.

To automate these procedures, we adopt a method that consists on preparing a configuration file that includes node configurations, network topology for the experiment and the scenarios.

The system that we designed is based on these configuration files to build the experiment environment and perform the scenarios execution.

2.3 Design of the experiment execution support system

This section describes the design of the system which supports the experiment execution in the environment described until now. It is a system for automating each procedure of the experiment stated in section 2.2.

2.3.1 Design of the experimental environment management model

The environment resources may be shared between several users. In order to prevent interferences between the user's experiments, we separate system wide resources management from the user's allowed actions. And we define respectively two management levels, *facility* and *user*. In the following we describe each of these management levels. (The relation between the management levels is shown in figure 2.3.)

Facility: This management level considers the resources which the administrator of the facility should manage, and the functions to administer these resources. Such resources are the actual nodes and the VLAN IDs. And the following is the list of allowed actions on those resources in this level:

- Node and VLAN IDs assignment to the system users.
- Setup and configuration of switches.
- Power supply management of the nodes.
- System wide servers management.

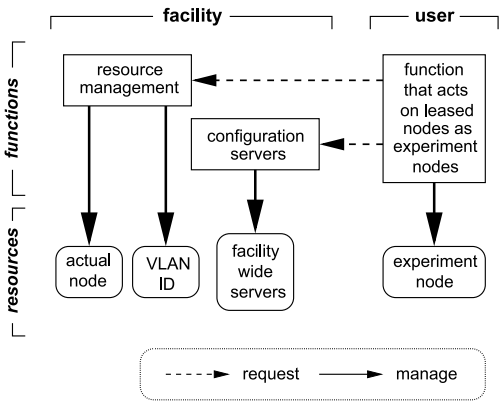

Fig. 2.3. The relation between two management levels

User: The leased nodes and the functions that customize their behaviors is considered on this level. and the functions that customize their behaviors. The following is the list of the allowed actions on these nodes:

- Loading of a configuration file.
- Installing the experiment software in the nodes.
- Executing experiment according to a scenario.
- Management of nodes status during experiment.
- Log files collection and analysis.

Generally, an administrator executes procedures in the facility level, and a user executes it in the user level. When a user wants to operate on some facility level resources, she/he need to execute a facility level function. In this case the facility level function verifies the access right to the requested resources.

2.3.2 Requirements for the system

The following are the requirements for this systems:

Loading of user configuration file Users have to write down the network topology of the experiment and informations to setup the

nodes as well as the execution scenario in a configuration file. And this file must be loaded by our system and processed to setup the environment and execute the experience scenario.

Nodes assignment to an experiment Nodes are assigned to users following the criteria described in the configuration files. In prevention of nodes which are in failure, users can acquire more nodes than described in the configuration file as spare nodes. The number of spare nodes is configurable.

Software settings of the experiment nodes Almost of the time we have the same OS and applications installed on nodes having the same role in the experiment. Therefore, disk images could be created for every role and installed in the node collection having the same role at the same time. In many cases we need a few disk images to setup all the experiment nodes. Moreover, the setup of IP addresses of each network interface is done according to the configuration file.

Building topology for the experiment The topology for an experiment is built by describing the desired topology in the configuration file. The topology is built virtually using VLANs.

Execution of an experiment according to the scenario To post the scenarios on each nodes, one way could manually perform a remote login followed by the scenario execution, another way could be remote execution using rsh[154] or ssh[311] etc. These manual procedures have many inconveniences, one of them is a synchronization problem, we can not control precisely the execution time of scenarios on each node. It becomes difficult to interpret the experiments results and outputs. Moreover, we cannot reproduce the same experiment because of the imprecision of the human factor.

There are some methods for driving on experiments automatically following the user's scenario. One method is to copy the scenario before the experiment starts, and each node will be programmed to execute the scenario independently at a specified timing. Another method is to use a scenario master: the scenario master transmits commands to the experiment nodes at a specified timing. Using the first method, experiment nodes can not synchronize with each other. By using the second method, the scenario master have to manage sessions for sending scenario to all experiment nodes. When there are many events at the same time, the management cost will be large and then the event timings might be delayed.

Therefore, our method uses scenario master only when some experiment nodes have to be synchronized. The method is to copy the scenario before the experiment starts, and generally each nodes execute the scenario independently. When the experiment node have to synchronize, it connects to the scenario master to mediate with other experiment nodes.

2.3.3 The functions which constitutes the execution support system

The system is designed so that the demand stated for the foregoing paragraph are fulfilled. And following is the list of the modules constituting our system.

TFTPd/DHCPd Each node boot up with PXE. TFTPd[262] and DHCPd are used when nodes boot up with PXE.

FTPd FTPd[218] is used as an interface to the file server.

Experiment Node Configuration Driver (ENCD) ENCD is the core function for driving experiments. ENCD loads configuration files and has a set of functions to execute the experiments.

Resource Manager (RM) RM manages the actual nodes and VLAN IDs. When ENCD requests resources to RM, RM decides suitable actual nodes according to number of

network interface or the type of network interface media described in the configuration file and then assigns the nodes to the ENCD. After node assignment, RM locks the nodes to prevent from assigning the same node to another ENCD. Moreover, it is able to manage defected nodes by avoiding their assignment to any experiment.

Node Initiator (NI) NI is used to setup the node software for the experiment. It works on nodes booted from the network and assimilated diskless system. It manages disk images installed on the node's hard disk partition by downloading these images from FTP server which address is provided by the ENCD server and then, saving the downloaded image in the local hard disk.

Facility Node Configuration Pilot (FNCP) When many experiments are running we have many ENCDs running, so the NI must distinguish the correct ENCD to communicate with. This way the user have to prepare several OS images with different IP address for ENCD. FNCP is used to avoid this problem: when the node boots it contacts the FNCP to obtain the IP address of the ENCD which manage this NI and starts communicating with it. FNCP is a facility management level service and its IP address is fixed. Using FNCP the same OS image can be used for all the experiments. So users are not required to make of OS images for each ENCD.

Wake on LAN (WoL) Agent In the environment we assumed, actual nodes boot-up using WoL. WoL use broadcast to send its magic packet. Sometimes there are multiple management segments. Since WoL magic packets don't go beyond local segment, a function to relay magic packets to remote management segments is required.

Directory Manipulator (DMAN) Experiment nodes load OS using the bootloader obtained from PXE. The bootloader is specified as a DHCP option. To change this DHCP option, DHCPd have to be restarted. DHCPd offers its service to all the actual nodes, so rebooting it could affect a large number of nodes. To avoid restarting DHCPd, we use symbolic links pointing to files specifying bootloader names for each node. By changing symbolic links, the bootloader also changes. Generally, users will have to change the boot partition of some nodes, but they cannot change the bootloader settings directly because TFTPd belongs to the facility level, DMAN is the interface to change this symbolic links to manage.

Table 2.1. Functions and their management levels

level	function
facility	TFTPd
	DHCPd
	FTPd
	RM
	FNCP
	WoL Agent
	DMAN
	SWConf
user	ENCD
	Scenario Slave
	NI

SWConf Switches also belongs to facility level, so a user can not setup VLANs directly. SWConf is a user interface to change switches configurations.

Scenario Slave Scenario Slave executes scenarios on experiment nodes. After introducing software into the nodes, the Scenario Slave communicates with ENCD and gets scenario for the node and then executed the scenario.

Each function belongs to a certain management level. The management level to which each function belongs is shown in table 2.1. Figure 2.4 shows the relationship of these functions when installing diskimage. In the figure, the functions connected with a solid line communicate each other. After installing diskimage, ENCD drives the user scenario by communicating with Slave

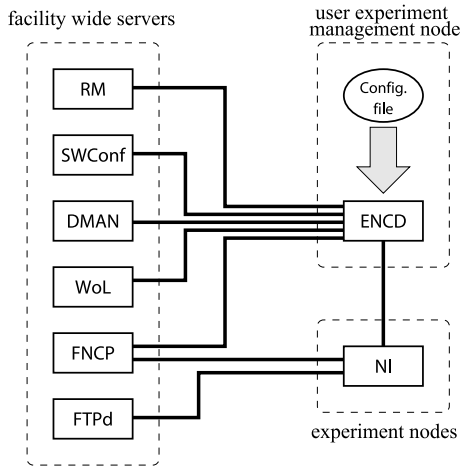

Fig. 2.4. Relationship of functions

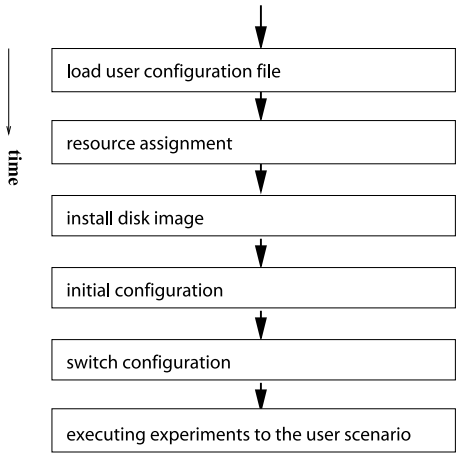

Fig. 2.5. The operation flow

on the experiments nodes according to the configuration file.

2.3.4 The operations flow

In this section, we describe the operation flow stated in proceeding section and in this section we explain these operations. Figure 2.5 shows running steps of our system.

1) Loading user configuration file The ENCD loads user configuration file, and recognize user's requirement for experimental network.

2) Resource assignment The ENCD communicates with RM to get actual nodes and VLAN IDs. In this time, ENCD can request more nodes than described in a configuration file. The RM tries to find required nodes that have desired characteristics. If RM can finds these nodes, it locks them and sends node list to the ENCD.

3) Installing disk image The ENCD states how the FNCP should redirect NI connection to it by adding a new entry in the FNCP containing its IP address and the NI IP address. The ENCD changes TFTPd's configuration via DMAN, this will setup the way leased nodes should boot as diskless system including NI. After that the ENCD sends request to the WoL agent to boot these leased nodes. When the leased node boots, the NI on that node communicates with the FNCP to obtain

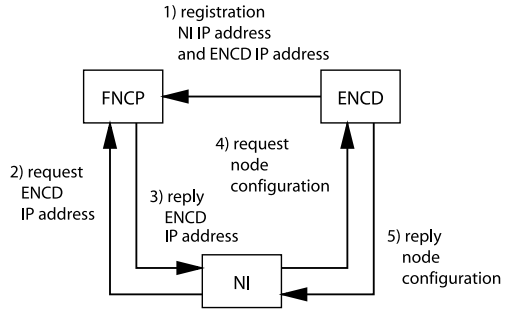

Fig. 2.6. Communications between NI, FNCP and ENCD

the IP address of the ENCD that will configure the node, after that, the NI communicates with ENCD directly. Figure 2.6 shows the communication between the NI, FNCP and ENCD after the node boots.

The NI downloads disk images specified by the ENCD from a FTPd which IP address is also specified by ENCD. The downloaded disk image is saved in one of the local node disk partitions, then the NI notifies the ENCD about these operations. The ENCD then changes the boot loader of the experiment node to boot from the partition where the new disk image was installed. Finally the node is rebooted by a request from ENCD to the NI.

4) Initial configurations When experiment nodes are booted with the desired disk image, the Scenario Slave runs and contacts the ENCD providing the MAC address of the

local node's network interfaces. The ENCD decides which IP address to attach to these MAC address and send this result to the Scenario Slave. The Scenario Slave then configure local network interface based on these informations. The user can perform additional settings of the nodes by specifying commands to be executed in the configuration file. Finally the Scenario Slave finishes the node initialization by notifying the ENCD.

5) **Switch Configuration** The ENCD configures the VLANs setting of the switches via SWConf.

6) **Executing experiments according to the user scenario** The Scenario Slave obtains the node's scenario from the ENCD, then executes it automatically. Only when it needs to synchronize with other nodes, the Scenario Slave communicates with the ENCD which is considered then as a global scenario master, and perform synchronization tasks between the experiment nodes.

These functions can be used to create disk images. The following is a description of disk image creation process. The user starts by installing the desired OS and applications in a model node, at this time the Scenario Slave must be installed as well. The user runs the ENCD specifying the model node IP address and the FTPd IP address. The ENCD changes the node's bootloader settings file located in the TFTPd using the DMAN. This change consists on setting the bootloader configurations to boot the node as diskless system loading a diskless OS with the NI tool installed. After rebooting the node, the ENCD asks the NI to upload the model disk image to the FTPd.

All the procedure mentioned above; disk image creation, experiment node setup, construction of the experiment topology and execution of the scenario are automated.

2.3.5 Writing the configuration file

The information and assumptions about the experiment environment is written in the configuration file which is loaded by our support system and processed to setup the node configurations, the network topology and execution of the experiment scenario. This section describes the structure of the configuration file.

Generally, nodes that have the same role in the experiment scenario have the same settings and configurations. Therefore, we adopt a syntax that allows the description of node collections, making the configuration file writing process easier when we have an important number of nodes.

We call *node class* an abstraction which defines nodes having common configuration and characteristics and we call *node set* a group of instances from the same node class. We also describe *network class* and *network set* the same way we described the node class and sets.

The experiment scenario is divided into *global scenario* executed by the management process and the *node scenario* performed by experiment nodes. A node scenario contains commands to be executed in the nodes and can contain instructions to send synchronization messages to the management process. The management process performs synchronization tasks according to the synchronization messages received from the experiment nodes. The node scenario is also considered as an attribute of the node class. We can control experiments with message passing and control statements like *if* or *for* loop in flexible. And we can use *sleep* sentence to stop the these nodes or scenario master's scenarios.

2.4 Implementation

We developed our support system that have functions we proposed. We design communication protocols used by our implementation except TFTP, FTP and DHCP. In this section, we describe the validation result of the our implementation and a simple example experiment.

2.4.1 Validation

We valid the our implementation using the

StarBED[274]. StarBED is an facility with a set of 512 nodes, a management network and an experiment network that have VLAN configurable switches.

A benchmark test with a set of 100 nodes are used for the validation, and we could obtain results confirming our expectations. Thereby, we could check that our system operates correctly and all the experient steps were successfully automated.

2.4.2 Example

We describe the flow of operations executed by our system when driving a simple experiment. The experiment is a benchmark test using netperf[273] program. Netperf program is used to measure the network throughput between a server and a client. Figure 2.7 shows message exchanges during the experiment. There is two experiment nodes and one management node, one of the experiment node is used as a netperf server and the anther one is used as netperf client.

After the scenario master delivers node scenarios to the slave nodes, these nodes execute their scenario. The experiment follows the steps mentioned below:

1. The server[0] executes netperf server: *netsever*.
2. The scenario master notifies the server[0] IP address to the client[0].
3. The client[0] generate traffic to the server[0] by executing netperf client program: *netperf*.
4. After executing the whole benchmark test, the client[0] sends the message 'cdone' to the scenario master.

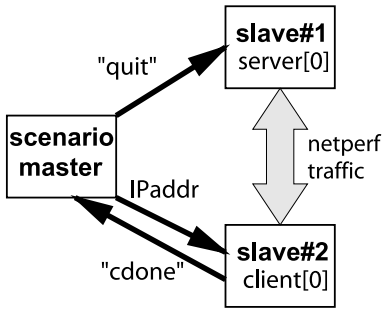

Fig. 2.7. Messages exchanged for the example experiment

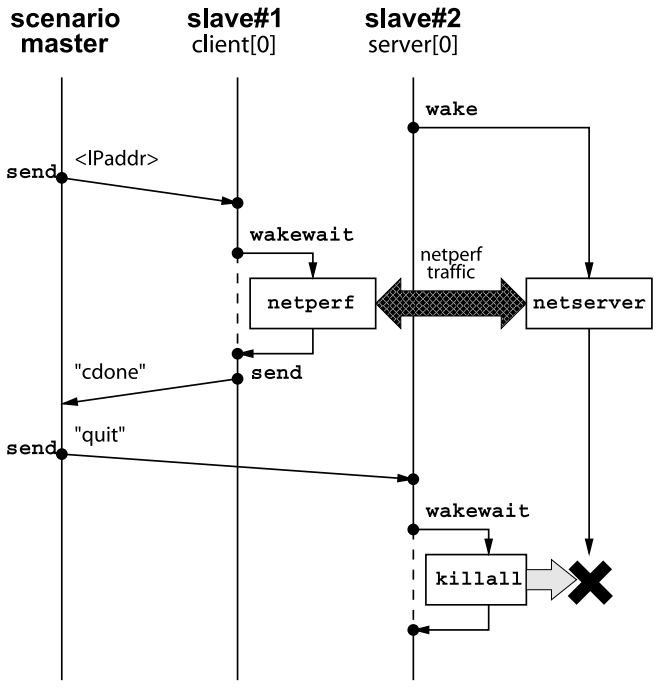

Fig. 2.8. Time line of the example experiment

```
 1: nodeclass svclass {
 2:  partition 2
 3:  ostype "FreeBSD"
 4:  diskimage "ftp://foo:passwd@172.16.1.1/s_image.gz"
 5:  netif media fastethernet
 6:  scenario {
 7:   wake "/sim/netserver" "/sim/netserver"
 8:   loop {
 9:    recv x
10:    msgswitch x {
11:    "quit" {
12:      wakewait "/usr/bin/killall" "killall" "netserver"
13:      exit
14:     }
15:    }
16:   }
17:  }
18: }
19:
20: nodeclass clclass {
21:  partition 2
22:  ostype "FreeBSD"
23:  diskimage "ftp://foo:passwd@172.16.1.1/c_image.gz"
24:  netif media fastethernet
25:  scenario {
26:   sleep 6
27:   recv dst
28:   sleep 24
29:   wakewait "/sim/netperf" "/sim/netperf" "-H" dst
30:   send "cdone"
31:  }
32: }
33:
34: netclass ethclass {
35:  media fastethernet
36:  ipaddrrange "192.168.3.0/24"
37: }
38:
39: nodeset client class clclass num 1
40: nodeset server class svclass num 1
41:
42: netset ethnet class ethclass num 1
43:
44: attach server.netif[0] ethnet
45: attach client.netif[0] ethnet
46:
47: scenario {
48:  send client[0] haddr(server[0].netif[0].ipaddr)
49:  sync {
50:   client[0] "cdone"
51:  }
52:  send server[0] "quit"
53:  exit
54: }
```

Fig. 2.9. Exsample experiment configuration file (main description)

5. When the scenario master receive the 'cdone' message, it sends a message 'quit' to server[0].
6. The server[0] kills the *netserv* process.

Figure 2.9 is an extract from the configuration file. Two node classes are defined from line 1–18 and from line 20–32, the first one is for the *netperf* server and the other one is for the client. In the configuration file we declared these classes by defining their attributes such as *partition*, *ostype* and *diskimage*. In both of these node classes, only one network interface is defined. We define one node per class. We can access node set members based on the node set name and the node number. For example, server[0] is a first node of the set 'server'. The networks are defined as the same way we define nodes. In the environment we are preparing for the experient we have a single network. The network class is defined at line 34 and the network set is defined at line 42. We can define the IP address space as an attribute of the network class. We use *attach* statement, to append a node's network interfaces to a network. The IP address is allocated automatically from the network's address space.

We also define three blocks describing the experiment scenario. Two of these scenario declaration are located in the node class declaration scope. And the dependent scenario block not attached to any class definition, is the global scenario. In the scenario block, there are some message control statements. A *recv* will wait a message and store the message strings to variable given as a parameter. The nodes can take different actions according to the received messages, this is stated in the configuration file by *msgswitch* sentence, there is an example the block from line 10. We can use *wake* and *wakewait* to run a command. When we use *wake* the command is executed as background process, and with *wakewait* the command is executed as foreground process. The *sync* sentence can be used only in the global scenario. It makes that the scenario waits for a message from a specified client. Figure 2.8 shows the time line of message transmission between scenario master and slaves.

2.5 Conclusion

Most realistic evaluation results for a system can be obtained by running the full-scale evaluation on the actual Internet. This approach could effectively used for some systems, but does not work for systems with certain characteristics. Examples of such systems are ones that generates massive traffic, ones that are fundemental services. Experimenting these systems on the actual Internet may heavily impact the activities on the Internet. The next best way of realistic simulation is to create an exact copy of the current Internet including people's activities and make changes to this copy, which is next to impossible because of the scale, complexity, and dynamic nature of the Internet.

However, it may be possible to simulate the actual Internet to some degree, using actual nodes with realistic topology and realistic traffic.

In this approach, we prepare a set of actual nodes and interconnection network facility to connect these nodes. By building experiment topologies virtually without changing physical connection, the cost of building can be dramatically reduced. And by sharing such environment among multiple users at the same time, the cost is also reduced. However, executing experiment in such environment, the user have to follow many steps to drive their experiments.

To simplify these steps, we designed a system which automate all these steps including building experiment topology using VLAN and executing the experiment following the user scenario. All the user have to do is describing the experiment topology and experiment scenario into a single configuration file.

We have implemented our system on the StarBED. The result of verification shows that our system worked as we expected. Moreover, the costs of a user to build an environment which the user desired and to execute the user's scenarios, is reduced.

Our future works will consist on the management of the nodes status, and feed back these status to scenario driving. With this function we can make an action when experiment nodes shows an ungeneral status. It is also needed that a log collection system and analysis assistance facilities. And preparing graphical user interfaces is important in order to make it easy for users to use our system.

Another major enhancement planned is the changes to scripting system. Internal review of the scripting and configuration language by potential StarBED users has revealed that everybody has different taste on how the language should look and work alike. From this result, we have decided to expose the API to internal scripting engine so that multiple language binding can be created. The change will also include ability to explicitly handle asynchronous events, which some reviewers have demanded.

第 3 章　SpringOS/VM：大規模ネットワークテストベッドにおける仮想機械運用技術

近年のインターネットへ導入される技術の対象範囲は大規模化しており、それにともない大規模なネットワーク検証環境が必要となってきている。実ノードを用い、実環境と同様の実装を用いた実験トポロジを構築できる大規模なテストベッドが提案されているが、要求される実験トポロジは日々大規模化しており対応は難しい。そこで、仮想ノード技術により実ノードを多重化し検証環境の規模を拡大する。

実ノードによる実験環境では実環境との同一性が問題になるため、汎用 OS を動作させることができる仮想機械を用い実ノードの多重化を行う。本論文では、我々が開発している実験支援システムである SpringOS に仮想機械を制御できるよう拡張を行った SpringOS/VM について述べる。

3.1 はじめに

近年、大規模なネットワークを対象とする技術が開発されるようになり、ネットワーク技術の検証に必要とされる実験トポロジも大規模化している。このような検証用ネットワークを構築する手法に、ソフトウェアシミュレータと実ノードによる実験環境がある。ソフトウェアシミュレータを用いて検証を行う場合は、ノード数が多ければ実験に必要な時間が長くなり現実的でない。実ノードを用いた場合は、実験シナリオに記述された時間で実験は終了するが、大規模な実験トポロジの制御は非常に困難である。しかし大規模なネットワーク実験の結果を、現実的な時間内で得るためには、実ノードを用いた実験環境を利用する必要がある。

実ノードを用いた大規模なテストベッドとして、StarBED[274] や Netbed[304] が提案されている。このような環境では基本的に多数の計算機を集め、1 計算機を実験トポロジ上の 1 ノードとして動作させることで大規模な実験トポロジを構築する。存在する計算機では実現できない規模の実験トポロジを構築するために、その都度、新たにノードを用意することは、経済的費用、設置するための空間、そして計算機の保守費用などさまざまなコストが必要となるため現実的ではない。

この解決策として、1 台の計算機を仮想的に多重化することで、さらに大規模な実験トポロジを構築する手法がある。すでに用意されている計算機を仮想的に多重化すれば、新たな計算機を用意するためのコストは発生せず、計算機の保守費用も増加することがないため、新たな計算機を用意する場合よりも、さまざまな点でコストが小さい。

1 台の実ノードを多重化するための手法は数多く発表されており、一般的に、仮想化の度合いが高ければ 1 台の計算機上で動作できる仮想ノードの数は増加する。しかしその一方、仮想化の度合いが高いほど、実環境との同一性が損なわれることが多い。

実ノードを用いたテストベッドでの実験は、実環境用の実装を大規模な実験トポロジ上で検証することが目的である場合が多い。したがって、実験遂行者が要求する OS やアプリケーションが変更なしに動作することは非常に重要である。汎用的な OS が動作しない場合は、実験遂行者が意図するアプリケーションの動作に支障をきたす場合や、実験遂行者が変更を加えた OS を利用できないなど、問題が生じる場合がある。

そこで我々は、汎用的な OS が変更なしに動作する仮想機械を利用し、テストベッドの更なる大規模化を目指す。我々は、大規模テストベッドでの実験支援システム SpringOS の開発を行っており、本論文では SpringOS を仮想機械を取り扱えるよう拡張した SpringOS/VM について述べる。

SpringOS/VM では、ある程度高速に動作するため多重化しやすい VMWare[293] を利用することとした。Netbed では jail[152] を利用した多重化を実現しているが、jail は FreeBSD のプロセス空間を分割するものであり汎用的でない。

3.2 実ノードによるテストベッドへの仮想機械の導入

本章では、実ノードによる大規模なネットワークテストベッドへ仮想機械を導入する際の問題点などについて述べる。

3.2.1 仮想機械の利用

前述の通り実ノードを利用したテストベッドでは、実環境との同一性は重要である。しかし仮想機械を利用した場合、ハードウェア部分をソフトウェアで模倣しているため同一性は低下する。

実験トポロジを構成するそれぞれの要素に必要な同一性の度合いは異なることが多い。一般的に主要な部分の同一性は高い必要があり、それ以外の部分は同一性が低くてもよい。実環境との同一性が高い必要性がある部分を実ノードで実現し、それ以外の部分を仮想機械で実現することにより、実験トポロジを大規模化できる。

ただし、実験遂行者は実験の性質を十分に吟味し、仮想機械を利用できるかどうか、また、利用できる場合は実験トポロジ中のどの位置に利用するかを決定する必要がある。この吟味が十分に行われない場合は、実験結果に影響をおよぼす可能性がある。

実験トポロジ中、仮想機械が利用できる部分の例を以下にあげる。

- 実験トポロジ中トラフィックを単に転送するだけのネットワークを構成するノード
- ユーザの動作を模倣し実験トラフィックを生成するノード

3.2.2 仮想機械利用の困難さ

大規模な実ノードによるテストベッドでは、実験遂行者のノード設定および実験トポロジ構築のための負荷を低減するため、支援システムが用意されていることが多い。このようなシステムを利用している実験遂行者にできるだけ仮想機械と実ノードの区別なく扱えることが望ましい。

我々が想定するテストベッドでは、多数のノードによる実験トポロジを容易に構築するために、VLANなどの仮想ネットワーク構築手法を用いて、物理接続を変更することなく実験トポロジを構築する。実ノードを利用する場合は、実ノードが接続されているノードのネットワークインターフェース（I/F）が収容されているスイッチの設定を行えば良いが、仮想機械を利用する場合は、仮想機械が動作している実ノードのI/Fを検索し、そのI/Fが接続されているスイッチの設定を行う必要がある。

また実験支援システムでは、実験遂行者が記述した手順（シナリオ）に従い自動的に実験を遂行する機能を持つことも多い。仮想機械を利用した場合は、仮想機械上で実行される実験駆動のためのシナリオと、仮想機械を動作させるための実ノード用のシナリオが必要となる。しかし、実験遂行者が混乱することを避けるため、仮想機械を動作させるためのシナリオは実験遂行者からは隠蔽することが望ましい。

以上のように、仮想機械を利用した場合は、仮想機械を動作させるノードの制御が必要となり、実験リソースの管理および実験手順が複雑化する。

我々は大規模なテストベッドで実ノードへのソフトウェアの導入および、実験トポロジの構築を支援するためのシステムとして、SpringOSを開発してきた。本章では、SpringOSの概要とSpringOSで仮想機械を扱うための拡張について述べる。

3.3 SpringOS/VM

SpringOSは実験支援システム全体の名称であり、さまざまな機能を持つモジュール群により構成されている。実験遂行者は各種設定を記述した1つのファイルを用意するだけで、SpringOSにより自動的に実験が遂行される。SpringOSの概要を図3.1に示す。

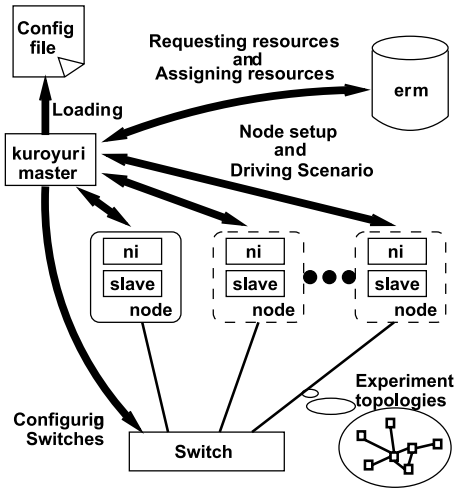

図 3.1. SpringOSの概要

本論文では、仮想機械を取り扱えるよう拡張したSpringOSをSpringOS/VMと呼ぶ。

3.3.1 SpringOS

SpringOSでは実験の実行主体であるkuroyuri masterが実験遂行者による設定ファイルを読み込んだ後、設定ファイルに従い他のモジュール群と連携し実験を遂行する。SpringOSの主要モジュールを表3.1に、処理手順を以下に示す。

1. kuroyuri masterによる実験遂行者の設定ファイルの読み込み
2. ermによるリソースの割り当て
3. ノードの起動とノード上でのni起動
4. niとkuroyuri masterによるノードへのOSお

よびアプリケーションの導入
5. ノード再起動
6. kuroyuri master から swconf にスイッチの設定リクエスト送信
7. スイッチ設定による実験トポロジ構築
8. kuroyuri master と kuroyuri slave によるシナリオ実行

我々は、テストベッドに実験用と管理用のネットワークが別々に用意されていることを想定しており、SpringOS による管理用の通信は管理用ネットワークを利用して行われる。これは、管理用の通信の実験への影響を防ぐためである。管理側のネットワークの IP アドレスは DHCP により、常に同じ IP アドレスが自動的に設定される。

SpringOS の詳細については、[186] に、実験ノードへのソフトウェア導入については、[329] にまとめられている。

表 3.1.　SpringOS の主要モジュール

要素	役割
erm	リソースの状態・属性管理 実験へのリソース割り当て
swconf	スイッチの設定
kuroyuri master	設定記述の認識 ノードへのソフトウェア導入指示 ノードへのシナリオ配布 シナリオ実行補助 他のモジュール制御
ni	ノードへのソフトウェア導入 （ディスクレス OS 上で動作）
kuroyuri slave	ノードでのシナリオ実行

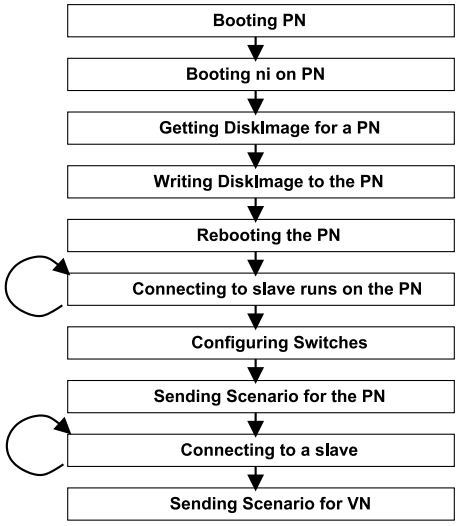

図 3.2.　SpringOS/VM の処理手順

3.3.2 SpringOS/VM の処理手順

仮想機械のノード設定の際の、処理手順を図 3.2 に示す。実ノードを実験用ノードとして利用する場合は、実ノードにシナリオが送信されれば実験が開始される。しかし、仮想機械を利用する場合は、実ノードに送信されるシナリオは仮想機械の設定および起動のためのものである。したがって、仮想機械設定、起動用のシナリオを実ノードに送信した後、仮想機械上で kuroyuri slave が起動するまで待機し、slave 起動後に仮想機械用のシナリオを送信する。

3.3.3 仮想機械を利用した場合のネットワーク構成

前述の通り SpringOS はテストベッドに管理用と実験用のネットワークが別々に用意されていることを想定している。同様に仮想機械も管理用および実験用のネットワークに接続することとする。仮想機械を利用した簡単な実験トポロジを図 3.3 に示す。図中の破線で示された長方形が実ノードであり、その中の実線の長方形が仮想機械を示す。実ノードと仮想機械にはそれぞれ I/F が用意されている。I/F の物理的な接続を実線で、仮想的な接続を破線で示した。今回は VMWare の vmnet-bridge を用い、仮想機械の I/F を、実ノードの I/F が接続されているネットワークに接続されているようにみせることができる。この機能を用い、仮想機械の各 I/F を実ノードが接続されているネットワークに接続する。

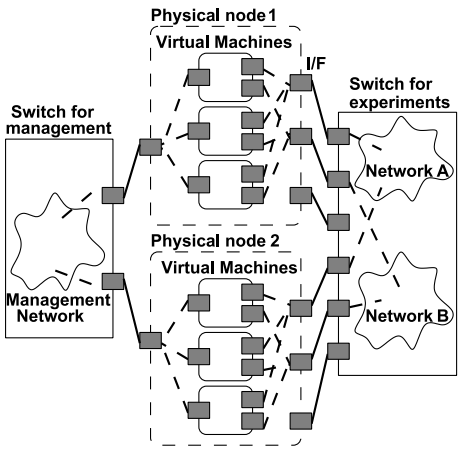

図 3.3.　仮想機械を用いた実験トポロジ

3.3.4 リソース管理

erm のデータベースには実ノードの情報が 1 台ずつ記述されている。仮想機械の情報については専用の記述方法を用意し、同じデータベースで管理することとした。図 3.4 に erm のリソース定義ファイルの一例を示す。これは実ノード physnode001 に関する定義である。このノードには I/F が合計 7 つあり、net からはじまる行の 2 カラム目の値で、その用途が管理用または実験用かを示している。manage が管理用であり experiment が実験用、そして empty はケーブルが接続されていないことを示す。またそのほかにも I/F の種類や、MAC アドレス、そして接続されているスイッチとそのポート番号などの情報が格納されている。仮想機械を扱うため、これらの情報に加え実ノードで起動し得る仮想機械の識別子を付加した。この識別子は、実ノードの設計上動作させることができる最大台数分を記述する。

仮想機械のリソース定義例を図 3.5 に示す。SpringOS は IP アドレスの設定などのために I/F の MAC アドレスを必要とする。VMWare は、MAC アドレスは指定された範囲内であれば自由に指定でき、VMWare の設定ファイルに記述することでこのアドレスを利用できる。この機能を利用し、前もって MAC アドレスを決定しこのアドレスを記述することとした。

管理ネットワークに接続されている実ノードの I/F には、DHCP サーバに MAC アドレスと IP アドレスが登録されており、起動時にその IP アドレスが自動設定される。仮想機械の管理ネットワークに接続されている I/F にも同様に常に同じ IP アドレスを設定するため、DHCP サーバに仮想機械で用いる MAC アドレスと IP アドレスの登録を行った。

3.3.5 ノード割り当て

動作させるアプリケーションの負荷を考慮し 1 台の実ノード上で動作させる仮想機械の台数を変更するため、実験遂行者が実験の設定記述に、実ノード上で起動する仮想機械の数を指定することとした。ノード割り当て時には、実験遂行者が指定した多重度と必要な仮想機械の総数から、必要な実ノード数を計算し、その値に予備ノード数を追加し erm に実ノードの割り当てを要求する。erm はこの情報に基づき実ノードを検索し、実験遂行者が指定した多重度で仮想機械を起動できる実ノードが必要数存在すれば、その実ノードを割り当てる。

3.3.6 実ノード上での仮想機械のための設定

erm により割り当てられた実ノードへのソフトウェアの導入の終了後、実ノード上では仮想機械のための設定を行うためのシナリオが実行される。このシナリオの主な役割は、vmnet-bridge の起動、VMWare の設定ファイルの生成、VMWare の起動である。

vmnet-bridge の起動 設定記述に従い、仮想機械の各 I/F と、実ノードの I/F の対応を調査し、そ

```
node physnode001 [
  bootdisk,SCSI
  net,manage,FastEthernet,00:00:4C:0F:77:C8,,172.16.2.1,"Linux""eth0""FreeBSD""fxp0"
  net,empty,FastEthernet,00:00:4C:0F:77:C9,,,"Linux""eth1""FreeBSD""fxp1"
  net,experiment,FastEthernet,00:C0:95:C4:52:70,"ethsw001,4/25",,"Linux""eth0"
  net,experiment,FastEthernet,00:C0:95:C4:52:71,"ethsw001,5/11",,"Linux""eth1"
  net,experiment,FastEthernet,00:C0:95:C4:52:72,"ethsw001,5/43",,"Linux""eth2"
  net,experiment,FastEthernet,00:C0:95:C4:52:73,"ethsw001,6/29",,"Linux""eth3"
  net,experiment,ATM,,"atmsw005,90",,"Linux""atm0"
  vnode,physnode001vm01,physnode001vm02,physnode001vm03,physnode001vm04, \\
    physnode001vm05,physnode001vm06,physnode001vm07,physnode001vm08, \\
    physnode001vm09,physnode001vm10
]
```

図 3.4. erm の実ノードリソース定義

```
vnode physnode001vm01 [
  physical,physnode001
  net,manage,FastEthernet,00:50:56:08:01:11,,10.8.1.17,"Linux""eth0""FreeBSD""fxp0"
  net,experiment,FastEthernet,00:50:56:08:01:12,"",,"Linux""eth1"
  net,experiment,FastEthernet,00:50:56:08:01:13,"",,"Linux""eth2"
  net,experiment,FastEthernet,00:50:56:08:01:14,"",,"Linux""eth3"
  net,experiment,FastEthernet,00:50:56:08:01:15,"",,"Linux""eth4"
  net,experiment,FastEthernet,00:50:56:08:01:16,"",,"Linux""eth5"
  net,experiment,FastEthernet,00:50:56:08:01:17,"",,"Linux""eth6"
  net,experiment,FastEthernet,00:50:56:08:01:18,"",,"Linux""eth7"
]
```

図 3.5. erm の仮想機械リソース定義

の結果をもとに vmnet-bridge を起動する。

VMWare 設定ファイル生成 VMWare の設定ファイルはテキスト形式であり、簡単に編集できる。すべての仮想機械に共通した設定部分を前もって用意しておき、このファイルに各仮想機械独自の設定を追加する。各仮想機械に独立な設定には、各 I/F の MAC アドレスおよび接続形式、およびディスクイメージのファイルなどが指定される。すべての I/F は vmnet-bridge の機能を用いて、実ノードが接続されているネットワークに接続する。

また、設定ファイルに従い、VMWare のディスクイメージをダウンロードする。

実験トポロジの構築 実ノードの I/F が収容されているスイッチの VLAN などを設定し、実験トポロジを生成する。

VMWare の実行 設定ファイルで指定された数の VMWare を起動する。

3.3.7 シナリオの実行

以上の手順により実ノード上で仮想機械が起動し、実験トポロジが構築される。この仮想機械上で起動する kuroyuri slave と kuroyuri master が通信を行うことで、SpringOS と同様にシナリオを実行する。

3.4 設定記述例と動作確認

我々は SpringOS にこれまで述べた拡張を行い、StarBED 上で netperf[273] を用いた実効帯域の計測を行った。実ノード上で動作する netperf のサーバプログラムである `netserv` へ、仮想機械上で動作する netperf のクライアントプログラム `netperf` からトラフィックを発生させるという簡単な実験である。`netperf` を動作させる仮想機械は、8 多重で動作させ合計 40 台用意したため、5 台の実ノードを利用することとなった。

各仮想機械の I/F は 2 つ用意し、それぞれを実ノードの別の I/F を通じ実験トポロジに接続した。各仮想機械では 2 つの `netperf` を起動し、それぞれ別の I/F からトラフィックを生成した。図 3.6 に実験トポロジを、図 3.7 と図 3.8 に設定記述の一部を示す。

図 3.7 は仮想機械の起動を行うための設定ファイルであり、実ノードの設定が記述されている。このクラスのインスタンスとして定義されるノードには 2 つ以上の I/F が用意されている。scenario からはじまるブロックに実行されるシナリオが記述されており、I/F の設定や、VMWare の設定ファイルを生成するためのスクリプトの取得と実行、そして VMWare の起動を行う。

図 3.8 は仮想機械の設定である。8 多重で仮想機械を起動するよう指定されており、仮想機械を起動するための実ノード設定には図 3.7 で定義された vmwareC クラスを利用する。このクラスのインスタンスの仮想機械は、2 つの I/F を持ち、1 つは実ノードの 0 番目の物理 I/F を通じ ethnet-0 というネットワークに接続され、もう 1 つは 1 番目の物理 I/F を通じ ethnet-1 に接続される。シナリオには I/F の設定後 kuroyuri master から `netserver` が起動しているノードの IP アドレスを取得し、そのアドレスに対して `netperf` を実行するよう記述されている。このクラスのインスタンスは nodeset ではじまる行で

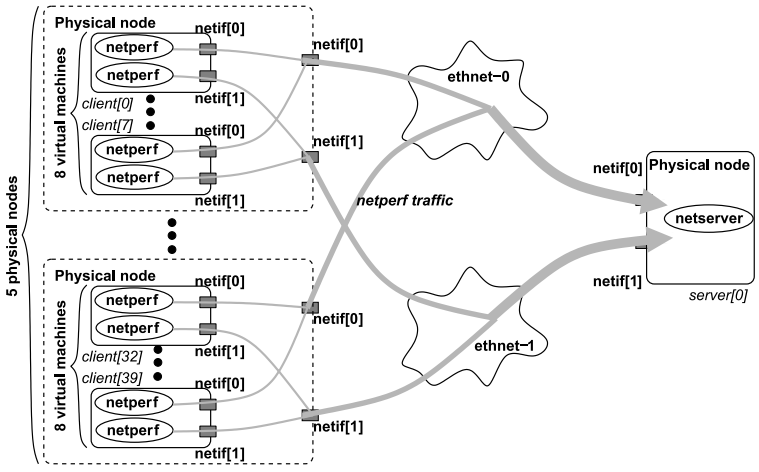

図 3.6.　netperf 実験用トポロジ

```
nodeclass vmwareC {
  method "HDD"
  disktype "IDE"
  partition 4
  ostype "Linux"
  diskimage "ftp://172.16.3.253/vmhost.gz"
  netif media fastethernet
  netif media fastethernet
  scenario {
    netiffit "/usr/local/bin/ifscan"
    wakewait "/sbin/ifconfig" "/sbin/ifconfig" self.netif[0].rname\\
      (haddr(self.netif[0].ipaddr))
    wakewait "/usr/bin/wget" "/usr/bin/wget" "-q" "ftp://172.16.3.253/vmsetup.pl"
    wakewait "/bin/mv" "/bin/mv" "vmsetup.pl" "/tmp/vmsetup.pl"
    wakewait "/usr/bin/perl" "/usr/bin/perl" "/tmp/vmsetup.pl" "/tmp/vm.hint"
    for(i=0;i<_launchvnodes;i++) {
      wake "/usr/bin/vmware" "/usr/bin/vmware" "-q" "-x"\\
        ("/root/vmware/linux0"+tostring(i)+"/linux.cfg")
    }
  }
}
```

図 3.7. VMWare 起動ノード用クラス

```
nodeclass clC {
  maxvnodes 8
  vntype vmwareC
  ostype "Linux"
  diskimage "ftp://172.16.3.253/linux.vmdk"
  netif media fastethernet via 0 net "ethnet-0"
  netif media fastethernet via 1 net "ethnet-1"
  scenario {
    netiffit "/usr/local/bin/ifsetup/src/ifscan"
    wakewait "/sbin/ifconfig" "/sbin/ifconfig" self.netif[0].rname\\
      (haddr(self.netif[0].ipaddr))
    recv dstA
    recv dstB
    wake "/usr/local/bin/netperf" "/usr/local/bin/netperf" "-H" dstA
    wakewait "/usr/local/bin/netperf" "/usr/local/bin/netperf" "-H" dstB
    send "cdone"
  }
}
nodeset client class clC num 40
nodeset server class svC num 1
```

図 3.8. 仮想機械用クラス

40 台用意することが指定されている。また、これとは別にサーバ用のクラスも定義されており、サーバ用のインスタンスとして実ノードが 1 台用意される。

図 3.7 の設定部分は実験遂行者が明示的に用意する必要はなく、別ファイルに用意されているため VMWare の利用をする場合は、図 3.8 のように vmwareC と利用すると記述すれば良い。しかし、実験遂行者が新たにクラスの定義を行えば、実験遂行者が意図する方法で VMware を起動することができる。また、新たなクラスを定義することで、別の仮想ノード実現手法にも対応することが可能であると考えている。シナリオ記述の詳細についても [186] にまとめられている。

これらの設定記述を用いて SpringOS/VM を実行することにより、40 台の仮想機械と 1 台の実ノードによる実験トポロジは問題なく構築され、netperf のトラフィックも発生した。これにより、SpringOS/VM は我々が意図した通り仮想機械を起動する実ノードの設定を行い、その上で仮想機械を設定記述通りに起動できていることが確認できた。

3.5 まとめ

既存の実ノードによるテストベッドの物理的な拡張は非常にコストが高く困難である。我々は、このようなテストベッドのノードを仮想機械を用いて多重化し、更に大規模な実験トポロジを構築するため、VMWare を制御可能な実験支援システム SpringOS/VM を実装した。

また、SpringOS/VM を StarBED で動作させ、実験遂行者による設定記述通りに実ノードおよび仮想機械を制御し、実験を遂行できることを確認した。

実ノードによるテストベッドの最大の利点は実環境と同様の実装が利用できる点である。実ノードを多重化し仮想ノードを生成した場合、この利点はある程度失われる。我々は実環境との同一性を重視し、汎用 OS を利用できる計算機レベルのエミュレータ

である VMWare を採用した。しかしこの他にも、さまざまな抽象度で仮想ノードを実現する手法がある。このような手法には、ユーザプロセスで OS をエミュレートする User Mode Linux（UML）[49] や Cooperative Linux（coLinux）[7]、実ノードで動作している OS のプロセス空間を分離する jail、仮想機械モニタ Xen[220] などがある。以上であげた仮想ノード実現技術では、汎用 OS を利用できないが、より高い抽象度で仮想ノードを実現するため、一般的に仮想機械よりも高速に動作する。したがって、どの仮想化技術を実験トポロジ中どの部分に利用できるかを厳密に吟味し、さまざまな仮想ノード実現技術を併用することにより、さらに大規模な実験トポロジを構築できる。

今後は、仮想ノードを利用できる場面について検討し、それを踏まえ、SpringOS をさまざまな仮想技術に対応できるよう拡張する。

第 4 章　不正アクセス等再現実験環境の統合手法に関する研究

インターネット全体の要素を考慮して根本的な解決を図るような新たなセキュリティ対策技術が求められているが、インターネット上で起こるセキュリティ事案は、規模拡大と複数事象の併発や相互干渉のため近年複雑化しており、原因の究明や対策の立案は、より困難になってきている。

このような状況に対し、我々は対策技術の検証基盤となる不正アクセス等に関する再現実験環境を研究開発してきた。しかし、事案の大規模・複雑化に対応するためには、単一の実験環境では限界がある。

そこで、本稿では、実験環境同士を連携させ、容易に規模の拡大や構成の変更に対処できるようにするための統合手法について述べる。

4.1 はじめに

インターネット上では、日々多くのセキュリティ事案が発生しており、枚挙に暇がない。これに対し、様々なセキュリティ対策が立案され、実施されているが、新しい攻撃手法との間でイタチごっこが繰り返されているのが現状である。そのため、根本的な解決を図り得るような新しい対策技術が求められている。

新たな対策の策定や対策技術の開発のためには、それらの有効性と悪影響の有無などを検証する必要がある。このような目的に利用するために、我々は検証基盤となる不正アクセス等に関する再現実験環境を研究開発 [318, 332] してきた。

再現実験環境では、事案に関連する各要素を実験環境内に模倣し、事案を再現することで、対策の有効性の検証や影響の計測を行うことができる。しかし、インターネット上のセキュリティ事案は、規模拡大と複数事象の併発や相互干渉のため近年複雑化しており、単一実験環境での再現は困難となってきている。

そこで、本稿では、インターネット上でのセキュリティについて、その近況と対策について概観し、再現実験環境の概要とその問題点を述べた上で、複数の再現実験環境の連携による大規模かつ複雑な事案の再現について提案し、我々の研究開発してきた再現実験環境を用いて統合手法について考察する。

4.2 背景

インターネット上でのセキュリティ事案の中で、特に影響の範囲が広いのは、通信帯域を浪費するような DDoS 攻撃と無差別に拡散・増殖するウィルスやワームである。

近年、大量の中継ホストを悪用する DDoS 攻撃はその規模を増しており、ウィルスやワームを中継ホストの乗っ取りに利用することで、より大規模な攻撃を可能としたものもある。

さらには、ある時点もしくはなんらかのトリガーによって自動的に攻撃を行うウィルスやワームもあり、総じてセキュリティ事案の影響範囲は広まる傾向にあるといえる。

4.2.1 セキュリティ対策

これらに対するセキュリティ対策としては、大きく分けて以下の 4 つが考えられる。

1. エンドホストへのセキュリティ対策の徹底
2. 組織単位の出入口での防御
3. プロバイダーなど網のレベルでの攻撃の排除
4. インターネット自身のセキュリティ能力の向上

セキュリティ事案に応じて、対策が必要な場合には、対策に関する情報に基づいてこれらの対策が随時実施される。

4.2.2 対策の策定と実験環境

このようなセキュリティ対策の実施は重要である。しかし、セキュリティ対策をより広い視点でとらえた場合、対策の実施は、最後の段階にあたり、そもそもの対策が策定されなければ、実施することはできない。実際に、新しい手法を用いた事案が発生するたびに、対策が策定され、その情報に基づいて対策は実施される。

4.2.2.1 対策の策定

対策の策定は、大まかに下記のような順で行われる。

1. 事案の分析による原因の究明
2. 対策手法の立案（修正）
3. 対策手法の効果の検証と影響の分析
4. 対策手法の確立と対策情報の周知

実際には、十分な効果が得られて、かつ、問題のないサービスなどになるべく影響を与えない手法が確立できるまで、2 と 3 を繰り返すことになる。

3 のような検証作業においては、セキュリティ上の深刻な影響をインターネット上の他のホストなどに及ぼしてしまう可能性があるため、インターネットに接続された通常のホストを用いるべきではない。しかし、検証や影響の分析においては、より通常のホストに近い状況が要求される。

そのため、これらの要求に応えるために、何らかの再現・模擬実験環境を用いるのが一般的である。

4.2.2.2 実験環境の種類と特徴

インターネット上のセキュリティに関する再現・模擬実験環境には、大きく分けて下記の 5 種類がある。

ネットワークシミュレーター ネットワーク上の各種の要素をノードとして抽象化し、モデルベースでシミュレートするもの。

ネットワークエミュレーター ネットワークシミュレータに、実際に通信を行うことができる機能を付加したもの。

PC やルータのエミュレーターで構成した実験環境 ネットワーク上の主要な要素であるホストとゲートウェイを、PC やルータのエミュレーターを用いて模倣する実験環境。

実 PC やルータで構成した実験環境 ネットワーク上の主要な要素を、実機で用意し、模擬する実験環境。

実際のインターネット上に構築した実験環境 実際のインターネットを用いて実験する実験環境。

詳細に関しては、関連研究（第 4.5 節）を参照して欲しい。

これらにはそれぞれ、再現の粒度や正確性などの再現能力（Reality）と耐規模性（Scalability）に大きな違いがある。一般に正確な再現をできる手法ほど耐規模性に乏しく、実験環境の運用が困難であり、抽象的な再現をする手法ほど、耐規模性に富み、実験環境の運用が容易である（図 4.1）。

では、セキュリティ事案の分析や対策の策定においては、どのような実験環境が求められるだろうか。

セキュリティ事案の多くは、OS やアプリケーションなどの実装に固有の脆弱性に基づいて引き起こされる。そのため、その原因の究明には特定の脆弱性

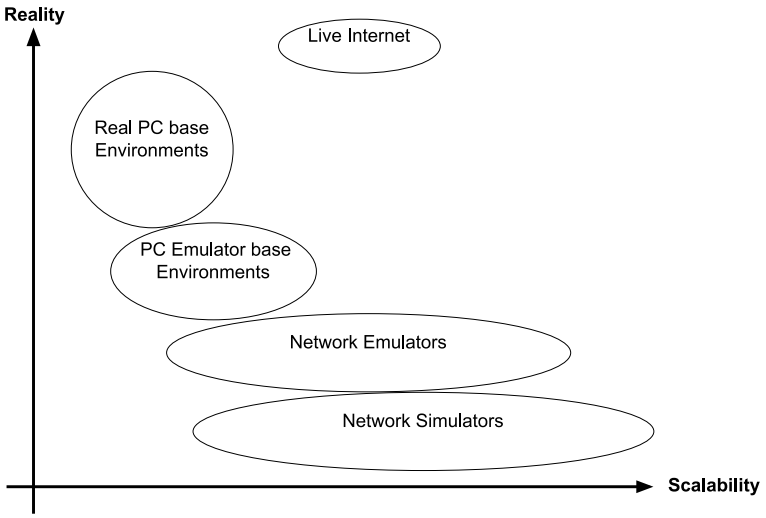

図 4.1. 各種実験環境の再現能力と耐規模性

を再現できる環境が必要となる。また、対策技術の権能や効果、影響を測るためには、その対策技術の実装を用いたコンフォーマンステストが必要となる。

同時に、近年セキュリティ事案の影響範囲は拡大しており、さらに対策技術も広範囲に適用することを前提としたものが登場しているため、これらの分析・検証には、大規模な実験環境が必要となる。

4.2.2.3 既存の実験環境の問題

このように、セキュリティに関する実験環境では、実装を用いた実験ができねばならない。すなわち、高い再現能力が要求される。しかし、セキュリティ事案の影響範囲の拡大や対策技術の適用範囲の拡大などから、耐規模性も要求されている。

実験環境には様々な種類があり、それぞれに得手不得手がある。また、その実装によって規模や構成は制約される。実験環境は、ある特定時期の事案の状況に合わせて構築されるのが一般的である。そのため、実験可能な規模や再現の対象は、設計時の状況に合わせて設計され、固定である。しかし、実際には事案の状況は変化しており、実験環境の構成を変更せねば対応できない。

ネットワークシミュレーターやエミュレーターであれば、規模の拡大は、内部のノード数の増加のみで対応できるため、特にコストを必要とせずある程度まで許容することができる。しかし、実機で構成された実験環境等の場合には、規模の拡大は、そのまま実際のノードの追加を意味することが多く、大きなコストを必要とする。

また、実機で構成された実験環境等では、再現対象の変更は、多くの場合、各ノードの OS やソフトウェアの変更で対応できる。しかし、ネットワークシミュレーターやエミュレーターの場合は、再現対象を変更する場合には、それ自体の実装を作り直す必要があり、求められる対象によっては再現することは非常に困難となる。

以上のように、単一の実験環境では、セキュリティ事案の状況の変化に対応することは困難である。そのため、事案の状況変化に合わせて大きなコストが必要となる。

4.3 実験環境の連携

前節で述べたように単一の実験環境では、その再現、模擬の能力は設計時に想定された限界がある。しかしながら、セキュリティ事案の状況の変化に合わせて、その限界を超えた能力が要求されるようになる。その際に、無尽蔵に大きなコストを費やすことができれば、状況に合わせた変更が可能であるが、実際には、コストは最小限に抑えることが要求される。

そこで本稿では、既に存在している実験環境同士を接続し、連携させることで、コストを最小限に抑え、規模の拡大や再現対象の複雑化に対応することを検討する。

4.3.1 複数の実験環境の連携

実験環境を連携させることで、たとえば以下のようなことが実現できると考えられる。

- 100 ノードと 500 ノードからなる実機による実験環境同士を接続して、全体で 600 ノードに規模を拡大する
- 実機とネットワークエミュレーターを連携させ、再現性能と耐規模性の両方の向上を図る

では、このようなことを実現するためには、連携で何を実現せねばならないだろうか。

4.3.2 連携における問題点と検討

まずは、実験環境同士が相互に影響を与え合えるようにするために、物理的、論理的に接続する必要がある。特に、インターネットセキュリティに関する実験に使う場合には、通信が相互にやりとりできる必要があると考えられる。また、通信だけではなく、計測・記録した情報についても、やりとりできる必要があるだろう。

さらには、それぞれの実験環境を何らかの方法で操作できなければならない。実験環境同士が地理的に離れた場所に設置されている場合には、何らかの遠隔操作手段が必要となると考えられる。

また、それぞれの環境がバラバラに動作するのではなく、イベント単位や時間単位で連動する必要がある。特に、通信に対する応答などは、互いに正しい順序で発生しなければならないと考えられる。

運用の容易さを考えれば、全体の実験環境管理や運用を統一的に扱える必要もあり、自動化なども含めて検討する必要がある。

このように、複数の実験環境を連携させる上で、接続の手法やどのように連携させるのか、実験環境間の様々な違いをどうするのかなど [331] 検討すべき問題がいくつかある。

そこで、次節では、実際の実験環境の連携事例を元に、検討することとする。

4.4 連携実験

このような実験環境の連携について検討するために、実際に我々が今まで研究開発してきた 3 つの実験環境の連携実験を事例として取り上げる。

4.4.1 各再現実験環境の概説

まず、各再現実験環境について、概要とその特徴について述べる。

4.4.1.1 SIOS 不正アクセス再現実験装置

SIOS 不正アクセス再現実験装置 [318] は、インターネットセキュリティを対象とした実機による実験環境である。これは SIOS[1] と我々が呼ぶシステムの一部である。東京都小金井市にある情報通信研究機構の小金井本部に設置されている。

不正アクセス再現実験装置は、100 台の PC からなる DDoS 攻撃再現部と帯域制御可能なインターネット部、各種の FireWall や IDS を備え、DNS/SMTP/Web などのサーバを擁する被害者部からなる。（図 4.2）

実際の攻撃ツールを利用して、最大 100 ノードからの DDoS 攻撃を模倣でき、実際の実装を用いて被害者への影響を模倣できる。

また、実験の管理や計測を行うためのエージェントが実装されており、実機による大規模な実験環境でありながら、詳細な実験管理を可能とし、運用の負担を軽減している。

4.4.1.2 VM Nebula

VM Nebula[332] は、PC エミュレータによる実験環境である。実機による実験環境の再現精度とエミュレータによる実験環境の柔軟性、耐規模性を兼ね備えることを一つの目標とした環境である。兵庫県神戸市にある情報通信研究機構の関西先端研究センターに設置されている。

VM Nebula は、PC エミュレータが動作する 4 台の模倣サーバとそれらを物理接続する 2 台のマルチレイヤスイッチからなる。（図 4.3）

サーバはすべて等価であり、2 台のマルチレイヤスイッチはそれぞれすべてのサーバへと接続しているため、各サーバには機能上の違いは無い。そのため、それぞれの役割は変更可能であり、かつ、それぞれの役割への機器の割り当ても任意である。

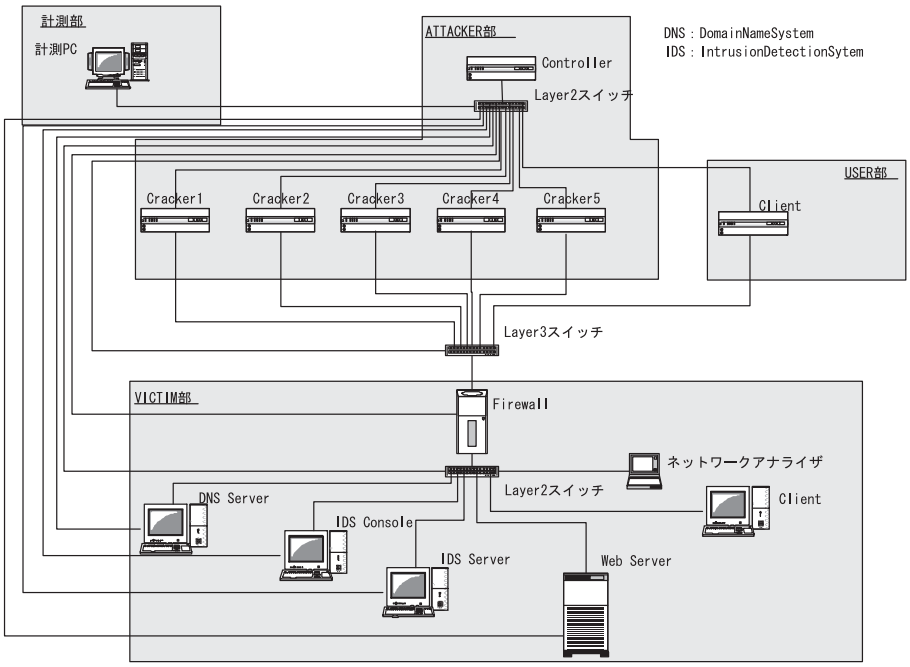

図 4.2. SIOS 不正アクセス再現実験装置

1 SIOS は Security Inteligent Operation Studio の略で、独立行政法人通信総合研究所（現情報通信研究機構）非常時通信グループが横河電機株式会社と共同で開発したシステムです。このシステムをもとに横河電機株式会社が独自に商品化したものが、商品としての SIOS であり、同社の登録商標となっています。

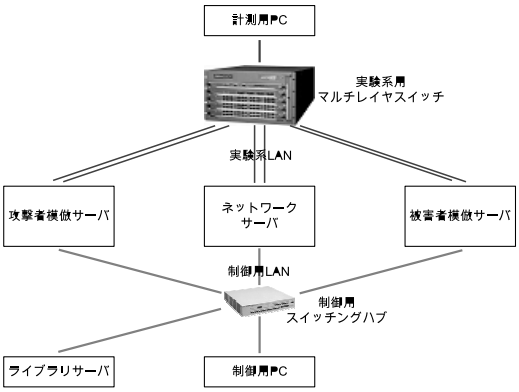

図 4.3.　VM Nebula

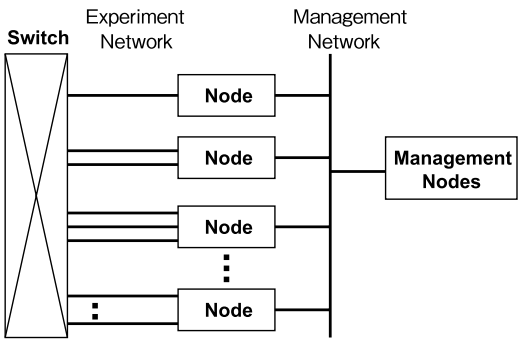

図 4.4.　StarBED

攻撃者や被害者の PC は、PC エミュレータを用いた仮想 PC によって模倣される。FireWall や IDS、各種のサーバからなる被害サイトは、複数台の仮想 PC によって模倣される。インターネットは、マルチレイヤスイッチを介して VLAN による接続と帯域制限を行うとともに、ルーティングソフトウェアを実行する仮想 PC を仮想ルータとして用いることで模倣する。

実際の OS 実装やサーバ実装をそのまま用いることができ、攻撃ツールなども PC 上で動作するものをそのまま利用することができる。

仮想 PC の構成やマルチレイヤスイッチの設定などを保存、配布する機能を持っているため、一度構成した実験系を再度構成することや再利用することが容易にできる特徴があり、頻繁に再実験を必要とするウィルスやワームの解析 [330] などにも用いることができる。

4.4.1.3 StarBED

StarBED は石川県能美郡の北陸 IT 研究開発支援センターの愛称で、大規模なネットワーク実験を行うための施設である。

StarBED は 512 台の PC とこれらを接続するスイッチ群で構成されている。それぞれの PC には 2 個以上のネットワークインターフェースが用意されており、実験用ネットワークと管理用ネットワークへそれぞれ接続されている。これは、実験用トラフィックと管理用トラフィックを分離するために必要な機能であり、これにより想定外のトラフィックによる実験結果への影響を防ぐことができる。管理側のネットワークには実験を支援するためのファイルサーバや DHCP サーバ、また我々が開発している実験環境の自動構築およびシナリオの自動遂行を実現するためのアプリケーションが動作するノードが設置されている（図 4.4 参照）。

基本的に各ノードやスイッチの物理的トポロジを変更せず、スイッチ群の設定を変更することで実験用トポロジを構築する。これにより、複数の利用者による同時利用や、実験用トポロジの構築のコストを軽減している。利用者は、各ノードのシミュレーション用のネットワークインタフェイスが収容されているスイッチの設定を変更することで、必要な実験トポロジを構築する。

実験支援ツールを利用することで、利用者は実験の設定記述を行うだけで、実験トポロジの構築および実験の遂行までが自動的に行われる。

4.4.2 連携手法の検討

執筆時点で、この連携実験に関しては準備中であり、接続後の状況に関しては機会があれば、別途報告したい。

これらの 3 つの実験環境は、互いに地理的に離れたところに設置されている。そのため、接続を行う物理回線が必要であり、また、相互に操作を実現するような何らかの遠隔操作手段が必要であった。

物理接続に関しては、専用回線のみによる接続、所内共用の広域 LAN と専用回線の組み合わせ、JGN2 を利用した接続の 3 つの案の中から JGN2 による物理接続を行うこととした。これは、主に以下の 3 点の理由による。

- 広帯域な（10 Gbps）ネットワーク接続を用意できること
- 実験専用線でインターネットとは別の隔離されたネットワークであること
- 3 つの実験環境がいずれも JGN2 のアクセスポ

イントに近く、低いコストで接続環境を準備できること

接続に際しては、JGN2 の「多地点同時接続サービス」による Ethernet 接続で 3 つの地点を結び、複数の VLAN を必要に応じて張り替えながら[2]利用する。東京都小金井市と石川県能美郡の間は、10 Gbps で、兵庫県神戸市との間は、1 Gbps で接続される。

各 VLAN の用途としては、以下を想定しており、余剰 1 本を含め、標準状態で 4 本の VLAN を用意している。

- 実験環境の運用、制御、計測情報用
- 相互の遠隔操作用
- 実験上のノード間の通信用

論理的な接続については、それぞれの実験環境で、プライベート IP アドレス空間を固有のルールで利用しているため、

- IP アドレスの割当
- ルーティングの有無、ルーティング方式

について取り決めを行う必要がある。また、各実験環境のどの部分とどの部分を接続するのかといったトポロジとその部分での出入りの方式なども検討を行わねばならない。

遠隔操作手段としては、まずは、各実験環境に既設の KVM over IP 装置があるため、これを利用し、それぞれの環境から実際に画面を見ながらキーボードやマウスを遠隔操作することとした。ただし、これでは運用管理が困難であるため、統一的なインタフェイスの用意などを実施する予定である。

なお、今回対象とする実験環境は、すべて、実際の OS 実装とソフトウェア実装を用いて実時間ベースで動作する。よって、今回は NTP による時刻同期は予定しているが、同期の手法には検討の余地がある。また、それ以外には今回は、これらを接続する上では、特別な仕掛けを用意することはしていない。

4.4.3 検証のための事案例の策定

連携手法が有効であるかどうかを測るためには、連携した実験環境上で何らかの事案を再現、模擬し、それが正しく実験できたか、容易に実験できたかなどを検証せねばならない。

そのためには、検証のための事案例が必要となる。事案例を考える上では、下記のような各実験環境の特徴を考慮する必要がある。

- 実際の攻撃ツールの準備と制御ができる SIOS 不正アクセス再現実験装置は、攻撃ノードや中継ノードの模倣に向いている
- ノード数が多く、ネットワークに関する設定能力に秀でた StarBED は、インターネットなどネットワーク部分の模倣に向いている
- 様々な実装を柔軟に変更することで特定の脆弱性を容易に再現できる VM Nebula は、被害ノードの模倣に向いている

このような特徴を考慮し、この 3 つの実験環境の連携に向いていると思われる事案例としては、「ある攻撃ツールを用いて特定の脆弱性を攻撃することで効果を上げるような DDoS 攻撃に対し、ある特定の IP トレースバック方式の有効性を検証する」といったものが考えられる。

しかし、検証を行うための事案例としては、向いているもの以外にも複数の例を用意し、実際にその権能を確認する必要がある。

4.5 関連研究

代表的なネットワークシミュレータとして、NS[292] がある。NS は、Tcl 言語によってノードとリンクの特性や動作を記述することで、主に IP ネットワークをシミュレートすることができる。既定の動作以外に関しては、C++ 言語で記述することで、独自に追加することが可能である。

IP ネットワークに関して、ルータやホストはノードとして高度に抽象化されているため、詳細な挙動に関しては、記述することもシミュレートすることもできない。

NSE[79] は、シミュレータである NS と実際の IP ネットワークとをつなげ、IP ネットワークのエミュレーションを実現する。

NS を元にしているため、ルータやホストは NSE 内部では高度に抽象化されているため、NS 同様に詳細な挙動に関しては、模倣できない。

Emulab[302] は、分散システムとネットワークに関する統合実験環境である。豊富な実験管理機能を持つのが特徴である。

実機とエミュレータの両方を利用可能な複合実験環境で、独自のディスクイメージ切替機構を持つ実 PC ノードと NSE を利用するエミュレータノード、それらをつなぐ物理ネットワーク接続から構成されている。

2　本来このような方式はサービスされておらず、今回特別にサービスして頂けることとなった。VLAN の張り替えは手動で行う。

物理ネットワークは、ソフトウェアによって変更可能なパッチパネルによって結線を変更でき、VLAN などと合わせて、柔軟にさまざまなネットワークを構成することが可能である。

また、複数の実験環境を RON[8] によってインターネットを介して接続し、より大規模な実験を可能にしている。

Emulab では、NSE を利用しているため、エミュレータノードでは通信に関する挙動は再現できるが、OS やソフトウェアの実装レベルでの挙動を再現できない。よって、正確な挙動を把握したい要素に関しては、実 PC ノードに配置する必要があるため、実 PC ノードの数がそのまま実験規模の制約になる点は、実機で構成した場合と同じである。ただし、通信に関する挙動だけが必要な要素をエミュレータノードに配置することで、規模の制約を緩和できる。

DETER testbed[45, 179] はセキュリティ関連の実験を行うためのインフラストラクチャである。DETER では Emulab による管理ツールを多く利用しているが、悪意のあるソフトウェアを動作させることを念頭においているため、セキュリティ的な観点から非常に頑強な設計がされている。現在は 3 サイトに分散した 100 台ほどの実ノードを用いて構築されている。DETER testbed では Emulab のツールが多く使われているため、Emulab と同様な特性をもっている。

4.6 考察と今後の課題

本稿では、単一の実験環境では困難な、事案の規模の拡大や複雑化への対応を、実験環境の連携によって可能とすることを提案してきた。ここでは、この手法が有効であるのかどうか、考察を加え、課題を述べることとする。

4.6.1 考察

まずは、耐規模性について考える。

複数の実験環境を連携させれば、単純に考えれば、それぞれの実験環境が耐えられる規模を合算しただけの規模に耐えられるように思われる。しかし、実際には制約となりうる要素がある。

例えば、実機による実験環境同士を連携させるためには、相互に何らかの通信環境で接続する。この接続に用いられる帯域幅によって、相互の通信は制約を受ける。そのため、耐えられる実験の規模も制約を受けることになる。また、シミュレーターと実機を何らかの変換機構で接続し、連携させる場合には、その変換機構の処理能力によって、規模は制約されることになるだろう。

このように、実験環境の連携によって対応できる実験の規模を拡大することはできても、その効果は連携に用いられる接続や変換機構の性能によって制約を受ける。

つぎに、複合化について考える。

実験環境の連携は、単純に連携させた実験環境の再現能力をそのまま複合化させることができるように思われるが、実際にはそうではない。

例えば、エミュレーターと実機による実験環境を連携させる場合、エミュレーターのインタフェイスの性能によって、相互の通信の粒度は制約を受ける。よって、再現能力も制約を受けることになる。また、実機による実験環境同士を連携させる場合でも、制御の粒度によって制約を受け、制御の粒度が低い方と同程度の再現能力しか発揮することはできない。

このように、実験環境の連携によって再現能力を向上させることはできても、その効果は連携に用いられる変換機構や実験制御の粒度によって制約を受ける。

よって、連携手法の性能は、接続や変換機構、実験制御の性能を如何に高められるか、もしくはそれらによる制約を如何に弱められるかによって評価することができるといえる。

4.6.2 今後の課題

連携実験に関しては、現在、準備中である。今後、ネットワーク接続の整備を進め、年度内には簡単な実験を実施できる予定である。

連携実験は準備中であるが、既に以下の課題がある。

すでに組み上げられている実験環境の実験管理は、それぞれの実験環境で閉じているため、連携に際して、他の実験環境の実験管理も行えるようにすることは容易ではない。特に、多くの場合、実験場の構築などは自動化されているが、複数の実験環境にまたがった実験場の構築を自動化することは容易ではない。そこで、どの部分を自動化するべきか、そうでないかを実験管理の工数などから導き、必要な部分のみ自動化を図るなど、手法については検討の余地がある。

複数の実験環境を連携させる上では、正しい実験

結果を得るために、イベント順の保護や時刻の同期が必要となる。特に、実機とシミュレーターやエミュレーターでは、時間の進み方に違いがあるため、何らかの方法で同期を取らねば、イベント順の入れ替えが起こるなど、正しい結果を得ることはできない。また、実機による実験環境同士では、NTP 等による時刻同期が考えられるが、完全な同期が図れるわけではないため、求められる同期の粒度によっては、十分ではない。このように、同期の手法は、実験環境の連携を考える上で、重要な課題の一つである。

無線などの別の物理メディアを用いる通信に関しては、既存の通信装置を前提とした実験環境では、模擬することが困難である。これに関しては、無線通信をエミュレーションする方法などが提案されている。また、無線通信に関する同様の実験環境があれば、それらとの連携を実現することで対応することも可能ではないかと考えている。

4.7 おわりに

本稿では、インターネット上でのセキュリティ事案に対する対策の策定を支援するための再現実験環境について、単一の実験環境の限界を述べ、複数の実験環境の連携によって、規模の拡大と複合化を実現することで対応することを提案した。

その上で、実際の実験環境を用いて連携手法について検討し、その課題について議論した。

現在、連携実験については準備中であり、年度内には実験を行い、報じられる予定である。

謝辞

本研究の研究費用は、科学技術振興機構の科学技術振興調整費によって賄われている。有用な研究費を与えて頂いた関係各位には、記して感謝の意を表したい。

また、本研究を推進するにあたって、北陸先端科学技術大学院大学の篠田研究室の各位には、様々な協力を頂いた。さらには、情報通信研究機構の北陸 IT 研究開発支援センターの各位には、実験環境の利用に際して多大な便宜を図って頂いた。この他、同機構の JGN2 関連部署の各位には、接続に関する無理な要求に応えて頂いた。記して感謝の意を表したい。

第 5 章　不正アクセス等再現実験環境の統合実験

インターネットセキュリティの再現実験環境では、事案に関連する各要素を実験環境内に模倣し、事案を再現することで、対策の有効性の検証や影響の計測を行うことができなければならない。しかし、インターネット上のセキュリティ事案は、規模拡大と複数事象の併発や相互干渉のため近年複雑化しており、再現実験環境は種類に応じて得手不得手があるため、単一実験環境での再現は困難となってきている。

そこで、我々は、既に存在している実験環境同士を接続し、連携させることで、コストを最小限に抑え、規模の拡大や再現対象の複雑化に対応することを検討する。

本稿では、このような実験環境の連携について検討するために、実際に我々が今まで研究開発してきた SIOS、VM Nebula、StarBED の 3 つの実験環境を JGNII を利用して接続した連携実験について、特に接続に関する評価を述べる。

5.1 はじめに

インターネット上では、日々多くのセキュリティ事案が発生しており、枚挙に暇がない。これに対し、様々なセキュリティ対策が立案され、実施されているが、新しい攻撃手法との間でイタチごっこが繰り返されているのが現状である。そのため、根本的な解決を図り得るような新しい対策技術が求められている。

新たな対策の策定や対策技術の開発のためには、それらの有効性と悪影響の有無などを検証する必要がある。このような目的に利用するために、我々は検証基盤となる不正アクセス等に関する再現実験環境を研究開発してきた。

再現実験環境は、その種類に応じて、再現の粒度や正確性などの再現能力と規模追従性に大きな違いがある。一般に正確な再現をできる環境ほど規模追従性に乏しく、実験環境の運用が困難であり、抽象的な再現をする環境ほど、規模追従性に富み、実験環境の運用が容易である。

セキュリティ事案の多くは、OS やアプリケーショ

ンなどの実装に固有の脆弱性に基づいて引き起こされる。そのため、その原因の究明には特定の脆弱性を再現できる環境が必要となる。また、対策技術の権能や効果、影響を検証するためには、その対策技術の実装を用いたコンフォーマンステストが必要となる。同時に、近年セキュリティ事案の影響範囲は拡大しており、さらに対策技術も広範囲に適用することを前提としたものが登場しているため、これらの分析・検証には、大規模な実験環境が必要となる。

このように、セキュリティに関する実験環境では、実装を用いた実験ができねばならない。すなわち、高い再現能力が要求される。しかし、セキュリティ事案の影響範囲の拡大や対策技術の適用範囲の拡大などから、規模追従性も要求されている。このような要求に対し、単一実験環境での再現は困難となってきている。

そこで、本稿では、既に存在している異なる複数の実験環境同士を接続し、連携させることで、再現能力や規模追従性の向上に伴うコストを最小限に抑え、規模の拡大や再現対象の複雑化に対応することを検討する。

このような実験環境の連携について検討するために、本稿では特に、実際に我々が今まで研究開発してきた SIOS、VM Nebula、StarBED の 3 つの実験環境を JGNII を利用して接続した統合実験について、特に VM Nebula と StarBED を接続した接続実験について述べる。

5.2 各再現実験環境の概説

まず、今回実験に利用した各再現実験環境について、概要とその特徴について述べる。

5.2.1 SIOS 不正アクセス再現実験装置

SIOS 不正アクセス再現実験装置 [318] は、インターネットセキュリティを対象とした実機による実験環境である。これは SIOS[3] と我々が呼ぶシステムの一部である。東京都小金井市にある情報通信研究機構の小金井本部に設置されている。

今回の実験では、SIOS は特に利用していないので、詳細は割愛する。

5.2.2 VM Nebula

VM Nebula[332] は、PC エミュレータによる実験環境である。実機による実験環境の再現精度とエミュレータによる実験環境の柔軟性、耐規模性を兼ね備えることを一つの目標とした環境である。兵庫県神戸市にある情報通信研究機構の関西先端研究センターに設置されている。

VM Nebula は、PC エミュレータが動作する 4 台の模倣サーバとそれらを物理接続する 2 台のマルチレイヤスイッチからなる。(図 5.1)

サーバはすべて等価であり、2 台のマルチレイヤスイッチはそれぞれすべてのサーバへと接続しているため、各サーバには機能上の違いは無い。そのため、それぞれの役割は変更可能であり、かつ、それぞれの役割への機器の割り当ても任意である。

攻撃者や被害者の PC は、PC エミュレータを用いた仮想 PC によって模倣される。FireWall や IDS、各種のサーバからなる被害サイトは、複数台の仮想 PC によって模倣される。インターネットは、マルチレイヤスイッチを介して VLAN による接続と帯域制限を行うとともに、ルーティングソフトウェアを実行する仮想 PC を仮想ルータとして用いることで模倣する。

実際の OS 実装やサーバ実装をそのまま用いることができ、攻撃ツールなども PC 上で動作するものをそのまま利用することができる。

仮想 PC の構成やマルチレイヤスイッチの設定などを保存、配布する機能を持っているため、一度構成した実験系を再度構成することや再利用すること

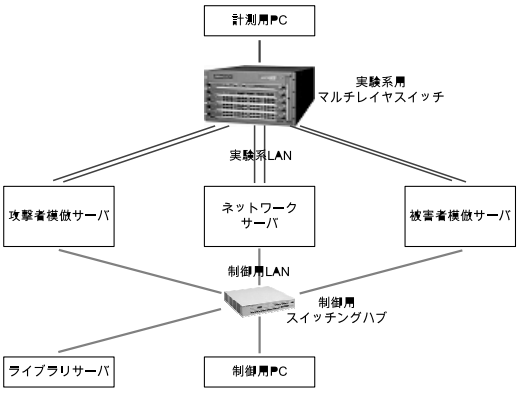

図 5.1.　VM Nebula

3　SIOS は Security Inteligent Operation Studio の略で、独立行政法人通信総合研究所（現情報通信研究機構）非常時通信グループが横河電機株式会社と共同で開発したシステムです。このシステムをもとに横河電機株式会社が独自に商品化したものが、商品としての SIOS であり、同社の登録商標となっています。

が容易にできる特徴があり、頻繁に再実験を必要とするウィルスやワームの解析 [330] などにも用いることができる。ただし、実験の遂行を自動化する仕組みは現在のところ提供されていない。

5.2.3 StarBED

StarBED[186] は石川県能美市にある情報通信研究機構の北陸 IT 研究開発支援センターの愛称で、大規模なネットワーク実験を行うための施設である。

StarBED は 512 台の PC とこれらを接続するスイッチ群で構成されている。それぞれの PC には 2 個以上のネットワークインターフェースが用意されており、実験用ネットワークと管理用ネットワークへそれぞれ接続されている。これは、実験用トラフィックと管理用トラフィックを分離するために必要な機能であり、これにより想定外のトラフィックによる実験結果への影響を防ぐことができる。管理側のネットワークには実験を支援するためのファイルサーバや DHCP サーバ、また我々が開発している実験環境の自動構築およびシナリオの自動遂行を実現するためのアプリケーションが動作するノードが設置されている。(図 5.2)

基本的に各ノードやスイッチの物理的トポロジを変更せず、スイッチ群の設定を変更することで実験用トポロジを構築する。これにより、複数の利用者による同時利用や、実験用トポロジの構築のコストを軽減している。利用者は、各ノードのシミュレーション用のネットワークインタフェイスが収容されているスイッチの設定を変更することで、必要な実験トポロジを構築する。

実験支援ツールを利用することで、利用者は実験の設定記述を行うだけで、実験トポロジの構築および実験の遂行までが自動的に行われる。

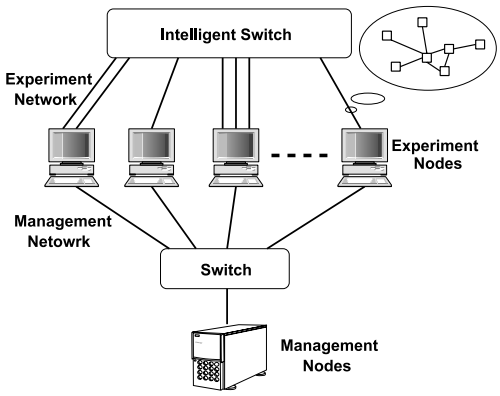

図 5.2. StarBED

5.3 統合実験

今回の実験は、統合の予備実験として、主に接続環境の性能に関して実験を行った。本節では、その実験について述べる。

5.3.1 接続環境

3 つの実験環境の物理接続には、JGNII を用いた。これは、主に以下の 3 点の理由による。

- 広帯域な (10 Gbps) ネットワーク接続を用意できること
- 実験専用線でインターネットとは別の隔離されたネットワークであること
- 3 つの実験環境がいずれも JGNII のアクセスポイントに近く、低いコストで接続環境を準備できること

接続に際しては、JGNII の「多地点同時接続サービス」による Ethernet 接続で 3 つの地点を結び、複数の VLAN を必要に応じて張り替えながら[4]利用する。東京都小金井市と石川県能美郡の間は、10 Gbps で、兵庫県神戸市との間は、1 Gbps で接続される。

各 VLAN の用途としては、以下を想定しており、余剰 1 本を含め、標準状態で 4 本の VLAN を用意している。

- 実験環境の運用、制御、計測情報用
- 相互の遠隔操作用
- 実験上のノード間の通信用

ルーティングを行わないレイヤ 2 の接続で、運用管理に利用する IP アドレスなどは調整する必要がある。

5.3.2 接続実験

今回の接続実験では、VM Nebula と StarBED を「実験環境の運用、制御、計測情報用」の VLAN を用いて接続した。接続図を図 5.3 に示す。

実験に利用したノードの NIC と OS を表 5.1 に示す。実験に利用した全てのノードには、同じネットワークに属するプライベート IP アドレスを付与した。

接続性能を測るために、今回は `ping` コマンドによる RTT の計測と、`netperf`[273] を用いた `TCP_STREAM`

4 本来このような方式はサービスされておらず、今回特別にサービスして頂けることとなった。VLAN の張り替えは手動で行う。

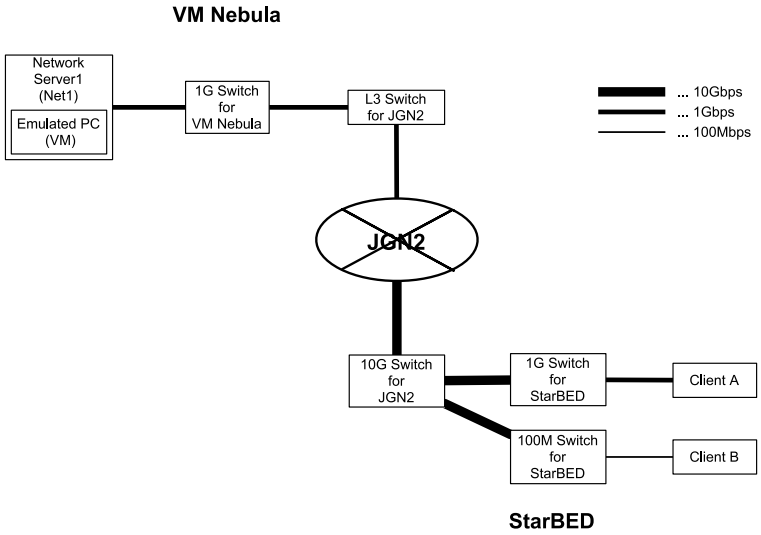

図 5.3.　VM Nebula と StarBED の接続図

表 5.1.　実験ノードの NIC と OS

環境	ノード	（略称）	NIC	OS
VM Nebula	ネットワークサーバ 1	(Net1)	1000Base-SX	RedHat Enterprise Linux AS 3.0
	仮想 PC ノード	(VM)	1000Base-SX	FreeBSD 5.3R
StarBED	Client A	(A)	1000Base-T	FreeBSD 5.3R
	Client B	(B)	100Base-TX	FreeBSD 4.4R

表 5.2.　実験結果

Sender	Receiver	TCP_STREAM	UDP_STREAM	RTT
VM Nebula intra				
Net1	VM	159.90	137.39	0.252
VM	Net1	298.90	152.30	0.252
VM Nebula → StarBED				
Net1	A	45.14	549.36	8.711
Net1	B	15.32	95.78	8.668
VM	A	28.40	152.63	10.580
VM	B	15.08	95.54	11.007
StarBED → VM Nebula				
A	Net1	24.25	412.70	8.754
A	VM	20.10	117.98	8.925
B	Net1	14.38	95.75	8.766
B	VM	14.28	93.24	8.747
StarBED intra				
A	B	93.24	95.88	0.228
B	A	93.38	95.93	0.211

と UDP_STREAMの性能について測定した。結果を表5.2に示す。なお、TCP_STREAM と UDP_STREAM はスループットで単位は Mbps（10^6 bits/sec）、RTT は ping コマンドで 100 回 ICMP を送信したときの平均RTTで単位は ms である。

表を見る限り、VM Nebula と StarBED 間の TCP_STREAM のスループットが最大約 12 分の 1 と異常に低い。特に、それぞれの実験環境内では、TCP_STREAM は UDP_STREAM と同程度かそれ以上であることから、JGNII を介した接続でのみ何らかの異常が発生[5]している可能性がある。

UDP_STREAM に着目してみた場合、1000Base のインターフェースで接続している Net1 と A の間で約 400〜550 Mbps 程度のスループットである。これは、各ノードの処理性能や JGNII を介した場合の限界性能であると考えられる。これに対し、100Base-TX のインターフェースで接続している B では、どのノードとの間でも約 90〜95 Mbps 程度のスループットが限界となっており、これはネットワークインターフェースによる限界と考えられる。VM と A との間では、約 100〜150 Mbps 程度のスループットがある。これは、VM Nebula の内部での性能と同程度であり、仮想 PC ソフトウェアやその実行サーバの性能による限界と考えられる。

5.4 考察

前節の実験結果を踏まえて、実験環境の連携にお

5　これに関しては、現在原因を究明中であるが、JGNII 内部でも当該実験時に異常なパケットが観測されていると連絡を受けた。

ける接続について考察を加えることとする。

実験環境を連携させる上では、様々な情報を流通させる必要がある。その中でも、インターネットセキュリティ事案の再現に最も重要なのは、事案の各要素（攻撃ホスト、仲介ホスト、被害ホストなど）を模倣する実験ノード同士の通信であると考えられる。なぜなら、インターネット上のセキュリティ事案は何らかの通信によって発生するため、その再現は、どんな通信によってどんな事象が引き起こされるのかを模倣することで行うからである。

複数の実験環境にまたがって実験ノード同士が通信する場合には、通信帯域や実際の性能（スループットや RTT など）が再現する通信に適しているかを把握しておく必要がある。

VM Nebula では、実際に実験ノードとして用いるのは全て仮想 PC ノードである。よって、VM Nebula と StarBED を連携させる場合には、VM Nebula の仮想 PC ノードと StarBED の各実験ノードを接続し、実験することとなる。そのため、留意すべきは、仮想 PC ノードと StarBED の各実験ノード間の性能である。

表を見る限り、`UDP_STREAM` のスループットは、仮想 PC ノードと StarBED の各ノード間では、VM Nebula 内部の性能と同程度か、StarBED 側ノードのネットワークインターフェースの性能を限界としている。StarBED 側ノードはネットワークインターフェースよって Class 分けされており、実験に必要なネットワークインターフェースを持ったノードを選択できる。よって、VM Nebula 内部の仮想 PC ノードの性能が、StarBED と連携した場合の接続性能の上限であることを踏まえて、各種の実験を構成する必要がある。実際には、複数の仮想 PC ノードを動かした場合には、今回の実験よりさらにスループットが低下することが予想されるため、StarBED との間で再現する要素の割当などを十分に検討して配分することが求められる。

また、RTT を見ると VM Nebula 内なら 1 ms 以下であるが、StarBED との間では約 8〜11 ms 程度あり、留意する必要があると考えられる。

5.5 おわりに

本稿では、不正アクセス等の再現実験環境の統合手法を検討するにあたって、我々が研究開発してきた VM Nebula と StarBED を用いて行った接続実験について述べ、考察を加えた。100 Mbps 程度のホスト間の通信の再現であれば、特に大きな問題を生じることなく大規模な再現実験が可能と考えられる。

今後は、SIOS を含めた実験を行い、10 Gbps の回線を使った性能評価や、実際に大規模な事案再現を試みるなど、統合による効果を検証していく予定である。

謝辞

本研究の研究費用は、科学技術振興機構の科学技術振興調整費によって賄われている。有用な研究費を与えて頂いた関係各位には、記して感謝の意を表したい。

第 6 章　まとめ

今年度は実験支援システムである SpringOS についての研究および、複数の実験環境を接続し、より柔軟な環境を構築するための研究を主に行った。

今後は異種/同種を含めた複数の実験施設を統合的に制御する枠組の構築に取り組み、世界的に広がりつつある PlanetLab との連携も検討して行く。また、実験や実験環境のモデリングなどについても検討を行う。