

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

Toshiyuki Miyachi[†]

Ken-ichi Chinen[†]

Yoichi Shinoda[§]

[†]School of Information Science / [§]Center for Information Science,
Japan Advanced Institute of Science and Technology
1-1, Asahidai, Tatsunokuchi, Nomi, Ishikawa, Japan

Abstract

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 somehow. 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.

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 im-

portance of evaluating new algorithms and their implementations has been increasing.

Software simulator is the most popular approach to evaluate them, and ns-2[1] 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[4] 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 perfor-

mance 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 difficulty 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[11] 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 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.

2.1 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[5]. 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 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 1 shows a space division of resource allocation considering three independent experiments, each set of nodes is completely independent from the others.

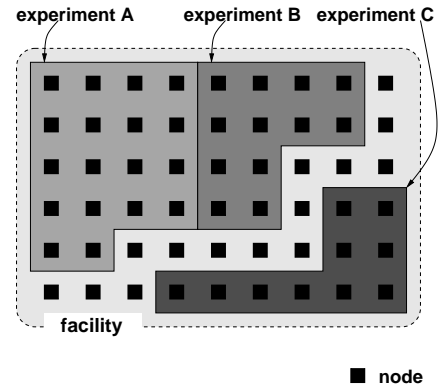


Figure 1. Using facility by space division

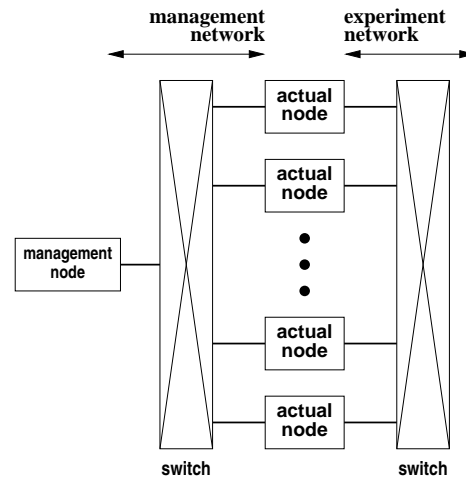


Figure 2. Concept of experimental environment

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 shows this concept.

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[7]. The bootloader specifies whether the node starts as diskless system or boots from local partition. The bootloader configurations are specified by the DHCP[6] 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.

3 Design of the experiment execution support system

This section describes the design of the system which supports the experiment execution in the environment de-

scribed until now. It is a system for automating each procedure of the experiment stated in section 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 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.

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.

3.2 Requirements for the system

The following are the requirements for this systems:

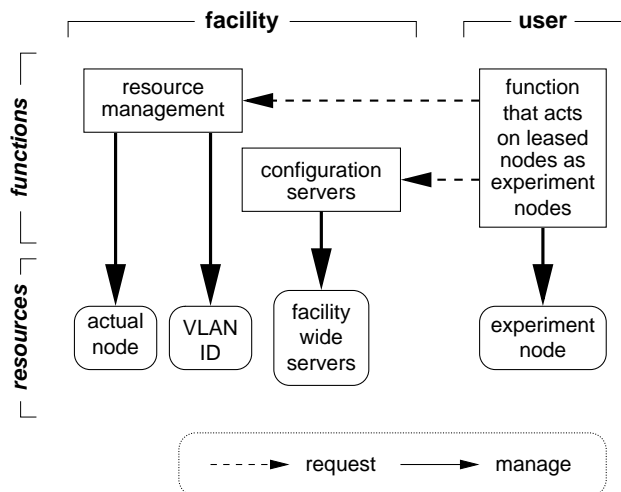


Figure 3. The relation between two management levels

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[8] or ssh[12] 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.

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[10] and DHCPd are used when nodes boot up with PXE.

FTPD FTPd[9] 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.

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

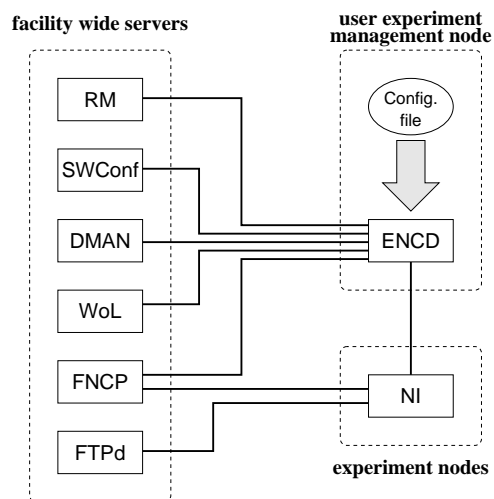


Figure 4. Relationship of functions

Table 1. Functions and their management levels

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

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 1. Figure 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 on the experiments nodes according to the configuration file.

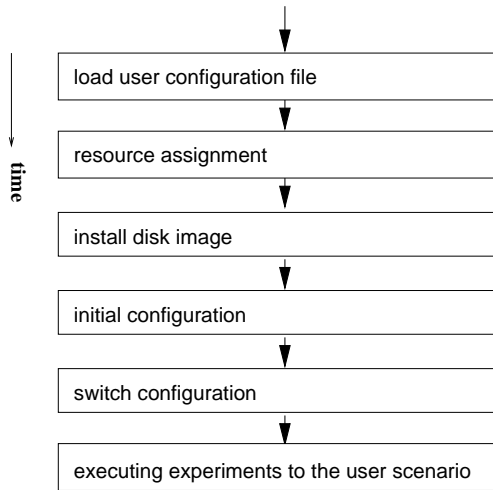


Figure 5. The operation flow

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 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 the IP address of the ENCD that will configure the node, after that, the NI communicates with ENCD directly. Figure 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

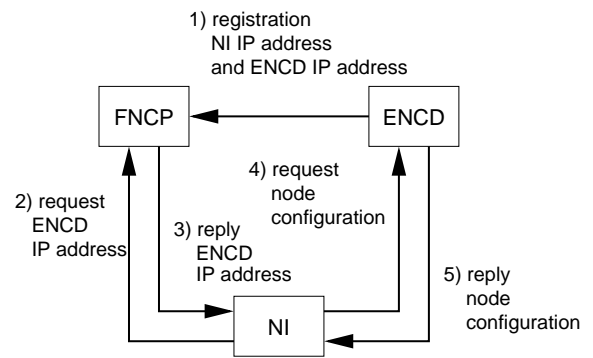


Figure 6. Communications between NI, FNCP and ENCD

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 FTP 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.

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.

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.

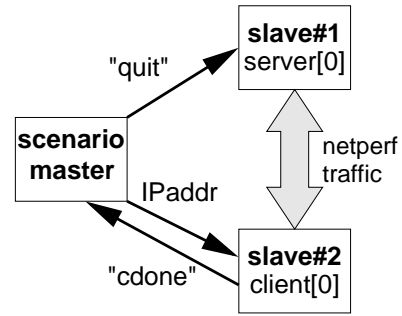


Figure 7. Messages exchanged for the example experiment

4.1 Validation

We valid the our implementation using the StarBED[2]. 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.

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[3] program. Netperf program is used to measure the network throughput between a server and a client. Figure 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.
5. When the scenario master receive the 'cdone' message, it sends a message 'quit' to server[0].

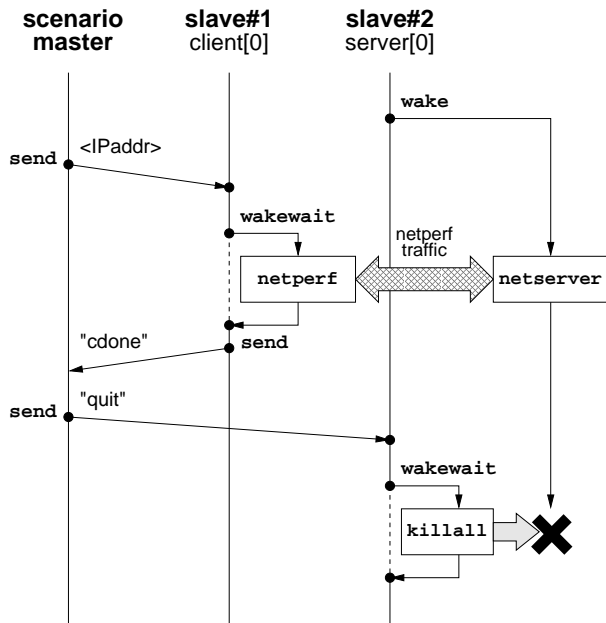


Figure 8. Time line of the example experiment

6. The server[0] kills the *netserver* process.

Figure 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 experiment 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*

```

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:   }
9:   loop {
10:    recv x
11:    msgswitch x {
12:      "quit" {
13:        wakewait "/usr/bin/killall" "killall" "netserver"
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:   ipaddr range "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: }

```

Figure 9. Example experiment configuration file(main description)

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 8 shows the time line of message transmission between scenario master and slaves.

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 fundamental 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.

References

- [1] *The ns Manual*.
- [2] The StarBED project. <http://www.starbed.org/>.
- [3] the public netperf homepage. <http://www.netperf.org/>.
- [4] Network emulation in the vint ns simulator. 12 1988.
- [5] *IEEE standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks*, 12 1998.
- [6] R. Droms. Dynamic Host Configuration Protocol, RFC2131. March 1997.
- [7] Intel Corporation. *Preboot Execution Environment (PXE) Specification Version 2.1*, 9 1990.
- [8] B. Kantor. BSD Rlogin, RFC1282. December 1991.
- [9] J. Postel and J. Reynolds. File Transfer Protocol, RFC959. October 1985.
- [10] K. Sollins. The TFTP Protocol (Revision 2), RFC1350. July 1992.
- [11] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. pages 255–270, Boston, MA, Dec. 2002.
- [12] T. Ylonen. SSH - secure login connections over the internet. Proceedings of the 6th Security Symposium (USENIX Association: Berkeley, CA):37, 1996.