StarBED and SpringOS Architectures and Their Performance

Toshiyuki Miyachi[1,3], Takeshi Nakagawa[2], Ken-ichi Chinen[3,1], Shinsuke Miwa[1,3], and Yoichi Shinoda[3,1]

[1] National Institute of Information and Communications Technology
Asahidai 2-12, Nomi, Ishikawa, Japan
[2] Fujitsu Hokuriku Systems Limited
Masuizumi 3-4-30, Kanazawa, Ishikawa, Japan
[3] Japan Advanced Institute of Science and Technology
Asahidai 1-1, Nomi, Ishikawa, Japan
miyachi@nict.go.jp

Abstract. StarBED is a network testbed begun in 2002 focused on running actual program code for software and hardware implementations. As hardware technologies develops, its facilities such as PCs and network equipments have been updated and we additionally developed SpringOS, a software support suite consisting of numerous program modules. The performance of StarBED and SpringOS increases as their equipments are upgraded and architecture is changed in terms of software and hardware. This paper gives an overview of the current StarBED architecture and SpringOS functions and then shows the results of performance evaluation as of October 2010.

1 Introduction

StarBED[1] is a large-scale general purpose network testbed and the SpringOS software suites supports experiments that take place on it. We have updated nodes and network equipments on StarBED and developed SpringOS, making it suitable for many types of experiments and raising the scale of experiment. StarBED now has about 1000 PC nodes only for network experiments and network equipments that connects PCs in order to build experimental networks and a management network.

To produce experimental contents and their schedules, it is important for users to know the fundamental performance of network testbed and supporting software. The performance of StarBED and SpringOS is changing during these update processes, so we periodically measure the performance of major functions. These values are also important for comprehending effects of hardware/software updates. The architecture of StarBED and SpringOS here is as of October 2010, and we provide measurement results.

T. Korakis et al. (Eds.): TridentCom 2011, LNICST 90, pp. 43–58, 2012.

Table 1. Node Specifications

		A	B	C	D	E	F	G1	G2	H
Model		NEC Express5800						Proside AmazeBlast neo920		HP ProLiant DL320 G5p
		110Rc-1	120Ra-1		110Rc-1		110Rg-1			
CPU		Pentium3 1GHz					Pentium4 3.2GHz	Opteron 2GHz		Xeon X3350
Memory		512MB					8GB	8GB	4GB	8GB
Disk		IDE 30GB		SCSI 36GB	IDE 30GB		SATA 80GB * 2	unavailable		SATA 160GB
Experimental NIC	FE	0	1	4	1	4	0	0	0	0
	GbE	1	0	0	0	0	4	1	1	2
Management NIC	FE	1	1	1	1	1	0	0	0	0
	GbE	0	0	0	0	0	1	1	1	1
Node no.		208	64	32	144	64	168	100	50	240
Introduced in		2002					2006	2007		2009

2 StarBED Architecture

StarBED, located in Ishikawa prefecture Japan, is funded by the National Institute of Information and Communications Technology(NICT). It is non-distributed type network testbed and all PC nodes described in this paper are at a single site.

First, we detail the StarBED physical topology.

2.1 Node Specifications

StarBED contains a total of 1070 PC nodes with differing specifications due to response to user requests and the timing of introduction. There are now nine node groups labeled A to H, with group G divided into G1 and G2[1]. The nodes have at least two network interface controllers (NICs), with one connected to the management network and the others connected to experimental networks. Table 1 shows the node specifications.

2.2 StarBED Topology

There are eight powerful network switches on the experimental networks, and users can build their network topology by configuring the VLAN on these switches without changing physical connections. The management network is usually configured statically, and users acquire a static IP address if they configure experimental nodes to use the DHCP protocol. This feature, which provides stable

[1] The nodes in group G were actually introduced by the Japan Advanced Institute of Science and Technology (JAIST) for its experiments, but when JAIST is not using them other researchers may use them.

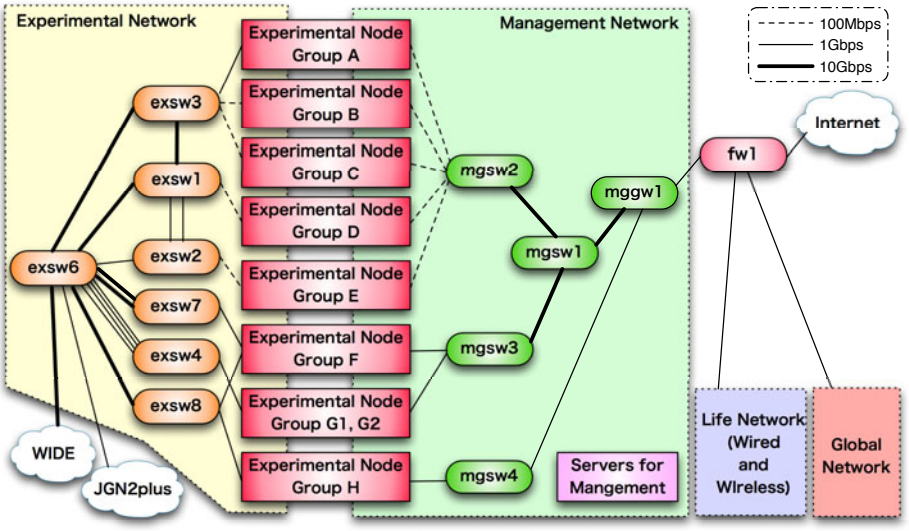

Fig. 1. Physical topology of StarBED

Table 2. Experimental Switches

Name	exsw1	exsw2	exsw3	exsw4	exsw6	exsw7	exsw8
Model	Cisco Catalyst 6509	Cisco Catalyst 6509	Foundry BigIron MG8	Foundry BigIron RX16	Foundry BigIron RX16	Foundry BigIron RX16	Foundry BigIron RX32
Exp. port number	144	256	400	150	0	168	984

access to experimental nodes, enables flexible construction of an experimental topology and configuration of PC nodes when the topology creation is still progress.

Figure 1 shows the physical topology of StarBED. The experimental network is drawn on the left side and the right side shows the management network.

The experimental network basically provides L2 topology. When users need routers to build their L3 topology, they should configure PC nodes as routers or bring their physical routers and connect them to the StarBED topology. There is only one experimental NIC for each group in Figure 1, but each node may in fact have more NICs, which is indicated in Table 1. In order to provide connections into other sites or introduce actual traffic into the experimental environment, two external lines are available via WIDE Internet[2] and JGN2plus[3]. Table 2 shows the models of experimental switches and number of ports connected to experimental nodes.

Table 3. Management Switches

Name	mgsw1	mgsw2	mgsw3	mgsw4	mggw1
Models	D-Link DGS3427	D-Link DGS3427 × 2 DGS3450 × 10 (Stacked)	D-Link DGS3427 × 1 DGS3450 × 10 (Stacked)	D-Link DGS3427 × 1 DGS3450 × 10 (Stacked)	Foundry NetIron MLX-4

The management network also has simple architecture but some switches are composed of several physical switches by utilizing stacking technology. Table 3 gives models of the management switches. Mgsw2, mgsw3 and mgsw4 are shown as a virtual single switch to mitigate management costs. There are several management servers in this network including DHCP servers, DNS servers, and servers for SpringOS. A firewall controls the access policies to/from the experimental networks, the management network, the life network for user's daily activities and the Internet. Through this firewall, StarBED environments for experiments are usually isolated from any other network for security purposes.

3 SpringOS Overview

SpringOS is a software suite for supporting network experiments primarily on StarBED[2]. Each of its component plays a role. SpringOS mainly supports following processes:

- Node and VLAN resource management
- Node power state control
- OS and application software installation
- (L2) Topology configuration
- Scenario execution

SpringOS configures and controls PC nodes, switches and management servers via the management network. Therefore its modules should be arranged on the management network in Figure 1. The behavior of SpringOS is detailed in Figure 2, and in the following we explain its main functions.

3.1 Node and VLAN Resource Management

The base function of SpringOS is management of experimental resources. The Experimental Resource Manager (ERM), which manages resource status, searches resources in its database and when it finds those that match a user's request, it allocates them and locks them so that they will not be allocated to other requests. These resources include experimental nodes and VLAN numbers which are needed to setup L2 topology.

[2] There are several examples in which it is introduced into a small physical testbed.

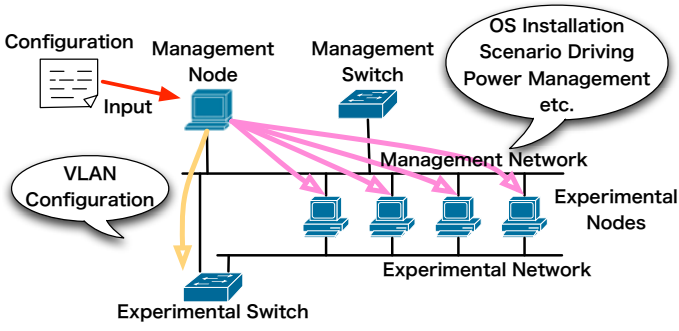

Fig. 2. SpringOS behavior

StarBED and SpringOS adopt two steps of resource allocation. The first step is to reserve StarBED facilities and the second step is to assign physical resources to logical resources, which are defined by users in the types of configuration files for SpringOS. The use of SpringOS is not mandated for users to conduct experiments other than the functions that support user mediation, which lets users use their original supporting software or that for other testbeds. Therefore we employ two steps of resource allocation. In the first step, users acquire resources with which they can perform any operation, such as OS reinstallation and setting up their own management environment. In the second step, using SpringOS to manage their experiments lets them allocate their logical topology definition onto the physical resources reserved in the first step only when users need this operation.

3.2 Node Power State Control

SpringOS enables remote power management using the existing technologies of WoL, IPMI, iLO, SNMP and executing UNIX commands on experimental nodes. The Power Manager (PWMG) daemon module await requests from users to control the nodes' power status. When it receives a request and the ERM confirms the user's ownership of specified nodes, it publishs messages to control the node statuses.

3.3 OS and Application Software Installation

SpringOS distributes a template disk image created from a template node into multiple experimental nodes in the OS and application software installation. Experimenters should install the OS and configure it as a template. The pickup command acquires a binary image of a partition or entire HDD and places it onto a file server. Then the wipeout command writes it to multiple nodes. This enables software installation, except different configurations for each node. The differences for each node may be installed on a part of a scenario execution or generated by the nodes themselves based on its management IP address acquired by a DHCP client.

```
rm 192.168.254.254 1234 ......... 1: IP address and port of ERM
rmuser starbeduser ............... 2: username for ERM
rmpasswd starbedpassword ......... 3: password for ERM
rmproject evaluation-10 .......... 4: project name for ERM
sm 192.168.254.254 1240 .......... 5: IP address and port of SWMG

activate a001 f001-168:0 ........ 6: activate ports
deactivate f001-168:1 ........... 7: deactivate ports
leavevlan 235 f001-168:1 ........ 8: delete ports from a VLAN
joinvlan 234 a001:0 ............. 9: add ports to VLAN
joinvlan 235 f001-168:0 ......... 10: add ports to VLAN
```

Fig. 3. Sample configuration of bswc.pl

When pickup or wipeout is used, a small diskless OS boots up on target nodes where a small client program is working. For an executed pickup, the client program reads the node's partition data or the entire HDD data and uploads it to a file server. For wipeout, it downloads a specified disk image and writes it into a specified partition or the entire HDD.

3.4 Topology Configuration

Experimental topology should be built by utilizing VLAN technology in StarBED. Users should acquire VLAN IDs from the ERM and request that the Switch Manager (SWMG) to configure the VLANs. This request includes VLAN IDs and port numbers of experimental nodes. When the SWMG receives the request, it asks the ERM for ownership of the resources and configures the experimental switches if the ERM confirms user ownership. A small client program, bswc.pl, for the SWMG is provided as a part of SpringOS and it communicates with the SWMG to build the L2 topology according to a user-written configuration file.

The bswc.pl command needs a simple configuration file as in Figure 3. Dashed lines and comments are added for explanation and must not be written into actual configurations. The first five lines are needed to access ERM and SWMG. The ERM will check ownership of the switch ports described in the following lines by using this user information. Lines 6 to 10 are actual configurations of switch ports. The SWMG recognizes a target switch port using an experimental node name and the node's port number that is connected to the switch port. In the configuration, *a001* means the first node of group A and *f168* means the 168th node of group F. Continuous nodes are defined as *f001-168* and the number that follows the colon "*:*" is the port number registered in the ERM database. *activate* and *deactivate* affect the indicated switch port, *joinvlan* and *leavevlan* indicate adding/deleting switch ports to/from a VLAN specified by the VLAN ID.

As mentioned, there are several experimental switches and users may designate configuration of VLANs distributed to multiple switches. The SWMG calculates the Inter Switch Link (ISL) configuration and configures these VLANs and switch ports.

3.5 Scenario Execution

Users should install the kuroyuri slave (slave) on experimental nodes when using a function for scenario driving. Basically, the SpringOS scenario looks like a UNIX command list, and the slave executes the scenario on an experimental node based on the scenario description sent from the kuroyuri master (master).

When slaves need to be synchronized, they send a message to the master. The master awaits messages from all slaves needing synchronization and upon receiving them sends another message to each slave to trigger the following scenario.

This function is also used for node setting and measuring node status. It enables users to execute any UNIX command on experimental nodes, so with this function users can configure the OS such as network configurations and upload log files including node or software status onto management servers.

The kuroyuri master can execute the overall steps of an experiment; acquiring experimental resources, writing a disk image into the allocated nodes, constructing experimental topology, and conducting the scenario. Meanwhile, each step can be executed by utilizing SpringOS client programs such as wipeout and bswc.pl, and the user can choose functions that satisfy their request for each situation.

When experimenters want to insert link characteristics such as delay and jitter between experimental nodes, they can use existing link emulator such as dummynet[4] on the experimental nodes. Especially for wireless link emulation on the wired network of StarBED, QOMET can help experimenters to build their environment[5].

The fundamental functions and architecture of SpringOS are described in another paper[1].

4 Evaluation Methods

We used the following functions to measure the performances of SpringOS on StarBED:

- OS and application software installation
- Topology configuration
- Scenario execution

To conduct experiments, these functions are important and basic on any network testbed and it takes a relatively long time to complete these roles.

This section shows the methods of performance evaluation for each function.

4.1 OS and Application Software Installation

As mentioned, SpringOS takes two steps for deploying the OS and application software onto experimental nodes: 1) create a disk image from a template node, then 2) distribute it to target experimental nodes.

In order to measure the time needed for these steps, we made a 20G partition on the group A, F and H nodes, and installed Fedora 13. We chose these groups because the group specifications of A, B, D and E are the same without experimental NICs, groups G1 and G2 have no HDD and group C has recently been used as management nodes. The OS is installed with the installer's default settings and we installed the kuroyuri slave for evaluating the driving scenario using these images. To compile the kuroyuri slave, we also installed gcc, zlib-devel, ncurses-devel, flex and bison.

The creation time and deployment time of the disk image may be influenced by the disk image size in terms of network traffic. SpringOS (pickup) normally uses the zlib library to compress the disk image before sending it to a file server. And the HDD status may interfere with the compression rate, so we tried to use the zerofree[6] command to fill the HDD sectors not used with "NULL".

Measurement A1. For measuring the time need to create template disk images using the pickup command, we executed the command with/without zerofree execution targeting the partition. Note the time should include the time uploading the disk image to a file server.

Measurement A2. To measure the required time to write the template disk image using wipeout, we distributed the disk image made by Measurement A1 with zerofree targeting for several sets of nodes: 1, 4, 16, 64, 128, and the maximum number of each group.

Measurement A3. To measure the writing rates of wipeout, we distributed three types of disk image made by Measurement A1 into a single node: one was compressed and zerofree was not executed, another was compressed and zerofree was executed, and the other was not compressed and zerofree was executed. The measured times show writing rates per 10 seconds.

Measurement A4. In order to observe the writing rates when wipeout targeting multiple nodes. We distributed the same disk images as in Measurement A3 to 128 nodes of group H.

4.2 Topology Configuration

To evaluate the necessary time to configure VLANs on the experimental switches, we executed bswc.pl to send a request to the SWMG. We created VLANs and added switch ports to them. The initial state of the experimental switch configuration was cleaned manually and there was no VLAN that was used for our evaluations.

We changed the parameters of configuring switch ports and VLANs as follows:

Measurement B1. For measuring the time needed to create one VLAN with multiple switch ports, we created one VLAN and added switch port to it and measured the time needed. The number of switch ports was changed to 1, 4, 16, 64 and 128 on group A.

```
nodeclass clclass {
  method "thru"
  scenario {
    send "setupdone" ................. 1: send message to master
    recv val ......................... 2: wait message from master
    callw "/bin/ping" "-c" "5" val .... 3: execute ping
    send "done" ...................... 4: send message to master
  }
}
nodeset client class clclass num 16 ... 5: node instance creation
```

Fig. 4. Scenario for slave

```
scenario {
  sync { ..............................
    multimsgmatch client "setupdone" .. 1: wait msg from all clients
  }
  multisend client "172.16.4.1" ....... 2: send target host of ping
  sync {
     multimsgmatch client "done" ...... 3: wait msg from all clients
  }
}
```

Fig. 5. Scenario for master

Measurement B2. To observe the required time to create VLANs that have only one switch port, we measured these time and and the number of VLAN-switch port pairs changes to 1, 4, 16, 64 and 128 on group A.

Measurement B3. As shown in Figure 1, group F node connections are divided among exsw7 and exsw8. We created a single VLAN and added switch ports to it on group F using SWMG ISL configuration.

4.3 Scenario Execution

To clarify the granularity of a scenario execution, we performed a simple experiment. Parts of this scenario are described in Figures 4 and 5. Figure 4 shows the scenario for slaves on experimental nodes. When the node's configuration is finished and the slave is woken up, the master sends a scenario description for each slave. The slaves execute the scenario as they receive it. In this scenario, the client will first send a "setupdone" message to the master. It then waits for a message from its master. The message from master will be kept in a variable "val". In this case, the slave assumes that the message includes a ping target IP address and sends an ICMP request to the target. After finishing the command, it sends a "done" message.

Table 4. Experimental Servers for the Evaluation

Name	Management Server	FTP Server
Model	HP ProLiant DL 360 G5	HP ProLiant DL 380 G5
CPU	Xeon E5405	Xeon E5405
Memory	2GB	16 GB
OS	Solaris 10	Solaris 10

Table 5. Measurement A1 - Creation time and size of disk image

Group	without zerofree		with zerofree		Image size Ratio [%]
	Creation Time [sec]	Size [GB]	Creation Time [sec]	Size [GB]	
A	6821	9.1	3551	0.760	8.35
F	3569	6.3	2687	0.875	13.89
H	2875	2.4	2539	0.870	36.25

Figure 5 shows the scenario description for the master. For the start of this scenario, it waits for a "setupdone" messages from each of the clients for verifying their status and then sends the IP address of the ping target to each node. It then waits for messages from all nodes to confirm whether the scenarios on the experimental nodes are finished.

Measurement C. For conducting the scenario on 1, 4, 16, 64, 128 and 150 nodes we measured the elapsed time to finish the scenario.

5 Evaluation Results

In these experimental environments server nodes were connected with mgsw1 as in Figure 1 and the specifications are shown in Table 4. SpringOS management modules such as ERM, SWMG, pickup, wipeout and kuroyuri master were operated on the Management Server. The FTP server was used as storage for disk images, so pickup uploads the disk image to the server and wipeout distributes it to experimental nodes.

5.1 OS and Application Software Installation

This section shows the results of the software installation.

Measurement A1. Table 5 indicates the result of Measurement A1. When the zerofree command was executed, the disk image size was reduced to one-third that of when it was not executed. FastEthernet is the management NIC of group A nodes and that of other groups is GigabitEthernet, which causes the

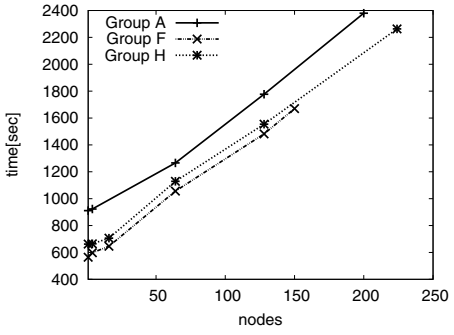

Fig. 6. Measurement A2 - Elapsed time to distribute disk images

gap in creation times. The image size ratio of the group A node is higher than for other groups, which may be caused by the size of the group A non-zerofreed disk image. The non-zerofreed HDD status should differ depending on the previous usage of the HDD, and there may be a larger amount of garbage in group A's disk than for other nodes. Group A's disk size is not so large and experimenters may use almost all parts of the HDD for their experiments, which causes a great deal of garbage, and this may be why the non-zerofreed compressed disk image is larget than that of other groups. The compression ratio of group H shows the worst value in the cases, but, the size of the zerofreed disk image is still one-third of non-zerofreed disk image, which shows that zerofree greatly impacts the disk image compression rate.

Measurement A2. Figure 6 designates the results of Measurement A2. The graphs appear to linearly increase as target nodes rise. However, tendencies in all graphs in the parts with under 16 nodes are gentle, so any bottleneck that existed was over 16 nodes. The node performance of group A and other nodes differ widely but the graphs tendencies are not so different, so this seems to be due to the FTP server-side problem, including NIC's capacity or the HDD read speed.

In the case of 128-node installation, the elapsed time was 1,777 seconds for group A nodes, 1,482 seconds for group F and 1,554 seconds for group H. Group H's performance was generally higher than group F, but some experimenters on StarBED have expressed that the group H disk performance is worse than group F. This worser performance of group H is demonstrated by these measurements.

Measurement A3. Figures 7, 8 and 9 show the results of Measurement A3. The graphs indicate the impact of compression and zerofree. Note that the graphs in Figures 7, 8, 9 and 10 are plotted every ten seconds without points because there are too many plot points to see the graph tendencies in detail.

In Figure 7, the tendencies of the default and zerofree graphs change after 3 GB of writing, in which actual Linux OS files exist, which means that the remaining parts of the disk image must be efficiently cleaned by zerofree. The

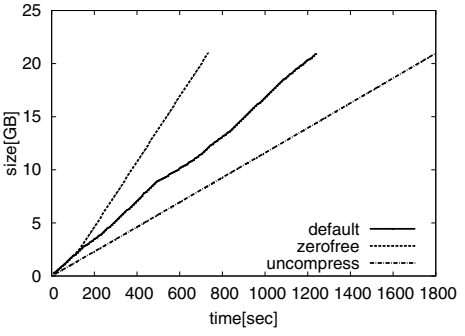

Fig. 7. Measurement A3 - Transition of writing rate (Group A, one node)

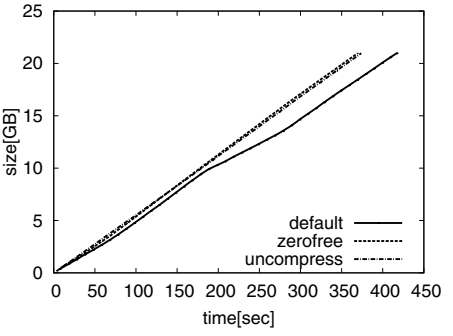

Fig. 8. Measurement A3 - Transition of writing rate (Group F, one node)

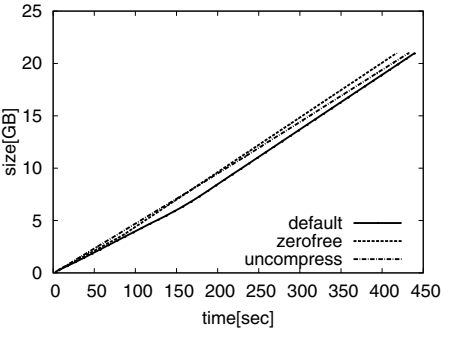

Fig. 9. Measurement A3 - Transition of writing rate (Group H, one node)

writing rate of the zerofreed one is the fastest, followed by the default that is non-zerofreed and compressed and the uncompressed is the slowest. This trend is likely caused by FastEthernet capacity.

Trends in Figures 8 and 9 are different from these in 7. The default shows the lowest performance, and the zerofree and uncompressed show similar tendencies. This is because the CPU resources to decompress the disk image were insufficient in comparison with the network bandwidth. The gaps of group H lines show little high performance than group F since CPU performance is higher than in group F. The group H graphs in Figure 8 and 9 are similar, which is caused by a bottleneck for group H on the FTP server; such as from the HDD read speed or NIC performance.

Measurement A4. Figure 10 illustrates the writing amount per 10 seconds. We selected one of 128 nodes to show the value. The graphs of other nodes for this experiment show the same tendency.

The zerofreed and uncompressed disk images are shown to be efficient as the number of target nodes increased due to bottlenecked network resources.

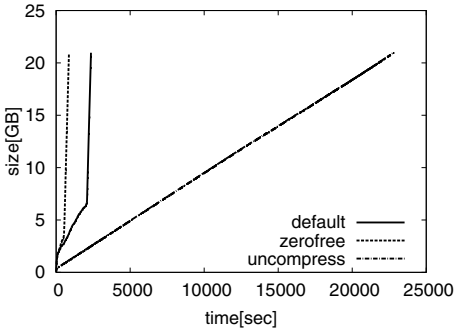

Fig. 10. Measurement A4 - Transition of writing rate (Group H, selected one node)

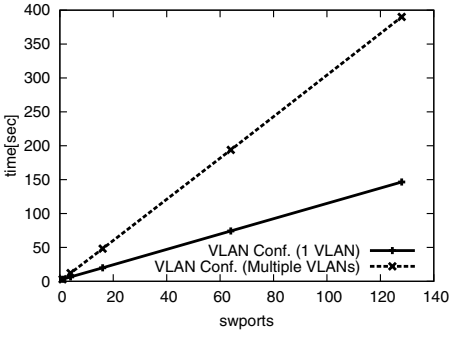

Fig. 11. Measurement B1 and B2 - VLAN configuration time (Group A)

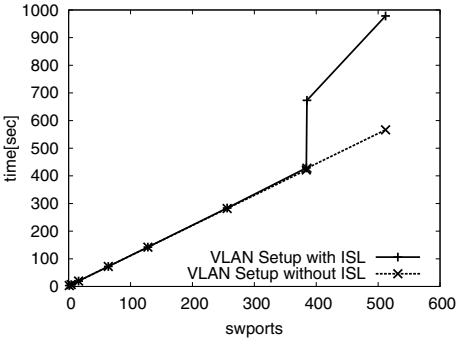

Fig. 12. Measurement B3 - VLAN configuration time (Group F, one VLAN)

The installation using zerofree finished in 863.9 seconds, the default was 2,355 seconds and umcompressed was 22,784.6 seconds.

5.2 Topology Configuration

The section explains the results of topology configurations.

Measurement B1 and B2. Figure 11 shows the results of Measurements B1 and B2. The values include the communication time between ERM and SWMG. This graphs clarifies that making many VLANs requires much more time than many switch ports in a single VLAN. The value in Figure 11 for configuring a single VLAN with 128 ports is 146.4 seconds and the time for 128 VLAN-port pares requires 390.1 seconds.

Measurement B3. Figure 12 illustrates the elapsed time for creating a single VLAN with multiple members. In this scenario for ISL evaluation, the first switch

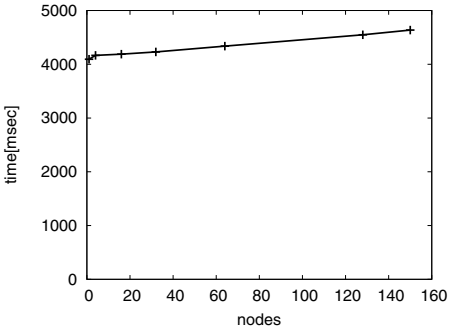

Fig. 13. Measurement C - Scenario driving (Group F)

(exsw8) has 384 ports for group F nodes and for greater configuration it should also configure another switch (exsw7). The gap in the figure is 244.5 seconds.

5.3 Scenario Execution

This section presents the results of scenario execution.

Measurement C. Figure 13 describes the results of Measurement C. The measured values show the total execution time of the scenario. The average time of ICMP RTT is 4,030 ms when we manually execute a ping command with the same options. Therefore we can see the management cost of SpringOS is quite low for this scale of experiment. But the value increases linearly so when the experimenter need to control a large number of nodes, it may influence experiments depending on their properties or purpose.

6 Discussion and Future Work

We evaluated SpringOS performance on StarBED in three fields: software installation, L2 topology configuration and scenario conducting. StarBED's PC nodes have been updated but node groups A-E have not been updated other than replacing the ATM NIC of group A nodes with GbE. These low-performance nodes are still effective as client nodes for experiments, but in terms of experimental environment construction, a great deal of time was needed for configuration for these nodes.

For software installation, we found that zerofree is effective in trimming the required time for creation and distribution of the disk image and storage spaces. The installation time per node in Measurement A2 is 13.9 seconds for group A, 11.6 seconds for group F and 12.1 seconds for group H. Considering the setup of each node including OS installation and manual software configuration, these numbers are quite low.

For VLAN configuration, creating 128 VLANs that have a single port took 146.4 seconds, or 1.14 seconds per port. Configuring VLAN-port pairs in Measurement B2 took 390.1 seconds for 128 pairs or 3.04 seconds for a single pair. These times are not so high but the tendency shows a tendency toward linear increase and setting up several thousands of switch ports may take a long time.

Measurement C clarifies that SpringOS has high granularity for controlling experimental nodes. The values increased by 500 ms for 150 nodes, compared to the value for a single node.

In these experiments we found many points that can be discussed for the future architecture of StarBED and SpringOS. Currently, SpringOS does not adopt multicasting for software installation or scenario driving. In network usage, multicasting is expected to be effective. Moreover, network boot technologies should curb installation time for some experiment types. However, the current SpringOS requires a maximum time of 13.9 seconds per node for group A, for which performance is lowest. This value is small enough for each node and we have to consider the impact of multicast and network boot on software installation. It is, however, important for users to select their preferred booting methods. Alongside developing SpringOS and introducing new technologies that can reduce resources involved in software installation, we should consider more facilitating greater usage of booting methods.

The results of disk image distribution for the group F and H nodes indicate a shortage of file server-side performance. In these experiments, we used a single FTP server, designated in Table 4. The file server which is provided to users for keeping their disk images and experimental data should have greater performance. We will reinforce servers, particulary in the aspects of HDD read speed and network capacity. This can be resolved by introducing a high-performance storage server or load-balancing system using several file servers.

In addition, the current SWMG needed several minutes to create complex topology requiring many VLAN and ISL configurations. The tendencies of the two graphs in Figure 12 show the same trend without ISL configuration, even when SWMG must configure two switches. Thus, there are some burdens by ISL configuration because the graph tendency with ISL configuration for 385 or more ports is different from with 384 or fewer ports. To decrease these added costs caused by ISL configuration, we will revise the configuration algorithms and publish command generation for each vendor's switch.

7 Conclusion

We conducted performance evaluation of SpringOS on StarBED. The results provide a yardstick for StarBED and SpringOS users to plan their experiments.

When we conducted evaluation in the same way in 2006, wipeout took 22,807 seconds for 200 nodes of group A using a 4 GB partition with the default settings. Now that management servers and switches were replaced, SpringOS was updated and the physical topology of StarBED was revised, the value with zerofree is only 2,380 seconds. Switch configuration also required much more time.

The results now show higher performance of StarBED and SpringOS for conducting experiments. As exhibited by, for instance, the roughly tenfold decrease in the required time for wipeout, we have proof that our update efforts for both software and hardware make StarBED and SpringOS have raised the level of performance. But the values also indicate areas in need of revision in future development.

The experiments for evaluating StarBED and SpringOS performance are not enough now to know actual bottlenecks and to seek efficient way to improve our architectures. In order to acquire these knowledge we should perform experiments with changing some elements in the experiments such as server configurations and physical topology. The StarBED nodes and physical topology will be updated in the spring of 2011, so we'll conduct more accurate experiments to clarify relations between architectures and performances after the update.

References

1. Miyachi, T., Chinen, K.-I., Shinoda, Y.: StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software. In: International Conference on Performance Evaluation Methodlogies and Tools, Valuetools 2006 (October 2006)
2. WIDE PROJECT Home Page, `http://www.wide.ad.jp/`
3. JGN2plus Official Web Site, `http://www.jgn.nict.go.jp/jgn2plus/english/index.html`
4. Rizzo, L.: Dummynet: a simple approach to the evaluation of network protocols. ACM Computer Communication Review 27(1), 31–41 (1997)
5. Beuran, R., Nguyen, L.T., Miyachi, T., Nakata, J., Chinen, K.-I., Tan, Y., Shinoda, Y.: QOMB: A Wireless Network Emulation Testbed. In: IEEE Global Communications Conference, GLOBECOM 2009 (2009)
6. Yorston, R.: Keeping filesystem images sparse, `http://intgat.tigress.co.uk/rmy/uml/index.html`