NAIST-IS-DT0461013

Doctoral Dissertation

Studies on Expeditious Preparation for Experiments in Network Emulation Testbeds

Mio Suzuki

Department of Information System
Graduate School of Information Science
Nara Institute of Science and Technology
February 15, 2010

Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Mio Suzuki

Thesis Committee: Suguru Yamaguchi, Professor
Minoru Ito, Professor
Yoichi Shinoda, Professor
Youki Kadobayashi, Associate Professor

Abstract

Before new technologies are introduced to a large-scale distributed system such as the Internet, they need to be evaluated practically to avoid any negative effects they may have upon existing systems. To perform such evaluations, many researchers and developers use a customized PC-based cluster also known as a Network Emulation Testbed (NET).

One of the greatest challenges in performing experiments on a NET is the time-consuming preparation needed for the user ' s experiments. Large-scale NETs, in particular, require efficient preparation within the one-week period that is typically reserved for a single user. Due to the large number of nodes, however, the preparation workload might be longer than it is on smaller ones. The origin of this problem can be considered in terms of four subproblems. The first problem is that topology configuration has no reusability. The second is the differences in assistant's tools and experimental procedures on each NET. The third problem is the lack of compatibility and complementarity of the assistant tools, and the last one is the difficulty of emulating a realistic topology.

In order to solve these subproblems, I propose AnyBed architecture in this dissertation. I divide the entire preparation process into three parts: network topology design, resource assignment, and node configuration. In the proposed architecture, I have three layers corresponding to each part: a design layer, an assignment layer, and an injection layer. Each layer consists of several components that are loosely coupled.

To solve the first subproblem, I divide network topology information for an experiment into two parts: logical network information and physical network information. Physical network information, which is stored in a physical network file, has NET-specific information such as hardware specifications for the nodes and wiring among network switches. Logical network information, on the other hand, which is recorded in a logical network file, contains experiment-specific layer three network topology. These

two information files are combined in such a way as to build an experimental network for the preparation. This division makes possible the reusability of network topology among NETs. This offers users the advantage of reduced user workload in such cases by reproducing the topologies of past experiments on another large scale NET, and moving that topology from real nodes to virtual nodes.

In regard to the second subproblem, I propose a common node configuration mechanism that is used on various NETs. This mechanism reduces the time it takes users to learn NET-specific procedures by enabling them to use common procedures to configure nodes and perform experiments among the NETs.

Concerning the third subproblem, I propose a layered modular architecture. This loosely coupled architecture facilitates an easy coordination with other assistant tools. AnyBed tools exchange data with other tools using simple Extensible Markup Language (XML) format or text format.

To resolve the last subproblem, I propose a method that can pick up and emulate the proper size of an Autonomous System level (AS-level) network from a public data set, as well as a method that emulates an actual operated Open Shortest Path First (OSPF) network topology on NETs. This method infers router configurations from the OSPF Link State Database (LSDB) obtained from one actual router on a backbone network, and then emulates the OSPF network topology with the inclusion of OSPF interface cost settings and assigned IP addresses on NETs. These two methods reduce the users ' workloads in the design of network topology in their experiments.

This proposed architecture has been implemented as an AnyBed toolset, which has been released as open-source software. From the evaluation results and the feedback of various users, I confirmed that AnyBed can expedite user's experiments.

Keywords:

Network Emulation Testbed, Experiment, Network Topology, BGP, OSPF

Acknowledgments

First, I would like to express my gratitude to my supervisor, professor Suguru Yamaguchi, for his helpful advice, and encouragement. I also deeply grateful to professor Minoru Ito for his support on my committee, and for giving me some relevant comments.

I am indebted to professor Yoichi Shinoda of Japan Advanced Institute of Science and Technology for his support of my work, and for providing me the study opportunity in StarBED.

I would especially thank to associate professor Youki Kadobayashi for his insightful comments and suggestions through discussions. His mind as a researcher had a great influence on me.

I also want to thank assistant professor Takesi Okuda, assistant professor Shigeru Kashihara, and assistant professor Hiroaki Hazeyama of Nara Institute of Science and Technology, assistant professor Katsuyoshi Iida of Tokyo Institute of Technology for their assistance. The discussions with them in our laboratory extraordinarily helped my work.

I appreciate the helpful feedback from heavy users of AnyBed, assist. prof. Hiroaki Hazeyama, Masatoshi Enomoto, Daisuke Miyamoto of National Institute of Information and Communications Technology. Their feedback was greatly useful for improving AnyBed.

I discussed issues on network emulation testbed and network topology emulation with Shinsuke Miwa, Toshiyuki Miyachi, Satoshi Ohta of National Institute of Information and Communications Technology, associate professor Ken-ichi Chinen of Japan Advanced Institute of Science and Technology, as well as, associate professor Kadobayashi and assistant professor Hazeyama many times. Through the numerous discussions with them, I could write this dissertation.

I would also like to thank Koji Nakao of Network Security Incident Response Group,

Information Security Research Center, in National Institute of Information and Communications Technology and members of his group. Their support in NICT was very helpful for me to write my dissertation.

Finally, I have had warm encouragement from IPLab members and my friends.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Before new technologies are introduced to a large-scale distributed system such as the Internet, they need to be evaluated practically in order to avoid any negative effects they may have upon existing systems. Such evaluations need a practical experimental environment that is similar to the actual Internet. To meet this requirement, many researchers use a Network Emulation Testbed (NET) to perform their experiments.

Firstly, I provide a brief description of NETs, and point out a number of problems one faces in expediting experiments on NETs. Next, I present the positive contributions offered by my work.

1.1 Network Emulation Testbed

Network Emulation Testbed (NET) is a customized PC-based cluster that enables users to perform network experiments by running their implementations on its nodes. Typical examples of experiments performed on NETs are deployment tests, verifications, and performance evaluations of software.

The NET is generally classified in terms of several axes: centralized/distributed, open access to the Internet/closed to local networks, and one node shared by multiple users/one node occupied by a single user. A centralized NET consists of commodity layer 2 switches and many PC nodes in one facility. Users can perform practical network experiments on an emulated network using these switches and PC nodes that act as routers. A distributed NET consists of many nodes that are located in universities and companies world-wide. In the case of an open access NET, users can perform

experiments with the nodes connected to the actual Internet. Some NETs, however, provide limited access to the Internet for each node out of concern for unnecessary traffic leakage. Some NETs adopt virtualization in order to share one node with multiple users, while others offer access on a reservation basis so that one user can occupy a part of all the nodes.

StarBED [1] and Emulab [2] are typical large scale centralized NETs. StarBED is a large scale centralized NET that is located in the Hokuriku Research Center at the National Institute of Information and Communications Technology(NICT HRC) [3]. StarBED is composed of 1070 PC nodes and several network switches for network simulation. Emulab is another large scale NET and the same name is used for its toolset. Located at the University of Utah, Emulab has 374 PCs, 40 wide-area nodes, 58 Wi-Fi nodes, and a front end to PlanetLab. Recently, NETs based on the Emulab toolset have been implemented in several countries. DETER [4] is the most well-known large scale NET based on the Emulab toolset. It focuses on experiments for research and development on cyber-security technologies. PlanetLab [5] is a distributed, live testbed that is available over the Internet. It has over 1000 nodes that are distributed at 494 sites around the world.

Preparations for experiments on these large scale NETs have been a tiresome process for experimenters due to the complexity of the physical resource assignment and the configuration overhead. To reduce this workload, StarBED, Emulab, and DETER have each produced NET assistant tools, which are customized for each environment. StarBED has produced SpringOS [1], Emulab has developed Emulab tools [2], and the DETER toolset is available in DETER Testbed [2]. These tools help users to set up the basic configuration of each node and to describe experimental scenarios in an NS-like manner [6].

Most NET-specific assistant tools of NETs focus mainly on resource assignment and basic configuration support such as address assignments for each interface and the setting of static routes on each node. A NET user therefore has to design a test topology and generate several configuration files. When a user wants to construct a large network topology with several routing daemons, for example, an inter Autonomous System (inter-AS) topology with a Border Gateway Protocol(BGP) [7] daemon for MOAS experiments [8], he must manually set up the configuration of each BGP daemon. Moreover, these NET specific assistant tools are not designed to be portable. The configuration files of these NET-specific assistant tools and the tools themselves are

currently customized for specific NETs or for the same structure testbeds. Due to this characteristic of NET-specific tools, a user cannot freely move his experiments among the various scales and characteristics of NETs.

1.2 Challenges in Expeditious Experiments

The goal of my study is expediting users' experiments on NETs. Every experiments has its time-consuming routine process such as preparation. I focus on this preparation. If this process will be shortened, the experiment becomes more efficient. To achieve this goal, I tackle the following problems about the preparation.

No reusability on network topology configuration

I assume that the NET dependence of network topology configurations can increase the workloads involved in reconfiguration. Configurations are difficult to reuse on different NETs because the configurations of a network topology typically include resource information that is dependent on specific software, hardware, and/or cable wiring on a NET.

Another reason for the difficulty of reusability is that most NET assistant tools are not designed for the generation of configuration files for a large network topology with several routing daemons. Users must generate their own configuration in addition to the configuration files generated by the assistant tools. These user-generated configurations often include dependent information.

Differences in assistant tools and experimental procedures on each NET

Users normally use a different assistant tool on each NET. If users were able to use the same assistant tools and procedures on various NETs, this would reduce the time it takes them to learn NET-specific tool and procedures.

Lack of compatibility and complementarity of assistant tools

Currently, performing an experiment simultaneously using multiple assistant tools is a troublesome process. The origin of this problem is the lack of modularity in the assistant tools.

Difficulties to emulate realistic Internet topology

Topology design for experiments is a time-consuming and difficult process for NET users. I explore an automatic method with which realistic Internet topologies can be emulated for the evaluation of Internet applications. These topologies contain inter-AS topologies only, intra-AS topologies only, and mixed topologies that include both.

I describe the details of these problems in Chapter 2.

1.3 Contributions

To achieve the goal of expediting users' experiments on NETs, my study offers the following contributions.

1.3.1 Proposed model for network emulation experiments

In order to expedite experiments on NETs, I propose my model of network emulation experiments, and the architecture and implementations proposed in this dissertation are based upon this model.

The scale of a user's experiment varies depending upon its purpose. Most users, however, gradually increase the scale of their experimental topology as their experiments progress. For example, in the case of networking software development, a user checks the performance of his software in one server. Next, he verifies the cooperative performance of several servers. Finally, he evaluates the software's scalability on many servers.

The problem is that there are NETs of various scales all over the world. Lab-level NETs normally have more than several servers. Large scale centralized NETs like StarBED and Emulab equip hundreds of servers. Users cannot use these large NETs on a casual basis because they require a reservation for every experiment.

My proposed model for network emulation experiments would enable users to perform their experiments with the same assistant tools on a NET that is suitable for the scale of their experiments. Adopting this model for the previously given example, a user would first use his development server for his experiment. Next, he would use a lab-level NET in his laboratory. Finally, he would perform his experiment using virtualized nodes on the lab-level NET and using real nodes on a large scale NET. In all these experiments he would use the same assistant tools.

1.3.2 Proposal of AnyBed architecture

On the basis of the model previously described, I propose AnyBed architecture.

In this approach, I divide the entire process of preparation into 3 parts: network topology design, resource assignment, and node configuration. In the proposed architecture, there are 3 layers corresponding to each part: a design layer, an assignment layer, and an injection layer. Each layer consists of several components that are loosely coupled.

To realize the reusability of network topology configurations, I divide the network topology information for an experiment into two parts: logical network information and physical network information. This division achieves a reusability of network topology among NETs, and it offers users the advantage of a reduced workload in such cases by reproducing the topologies of past experiments on other large scale NETs, and moving that topology from real nodes to virtual nodes. To enable common assistant tools and common experimental procedures to be used on each NET, I propose a common node configuration mechanism that can be used on various NETs. This mechanism reduces the time it takes users to learn NET-specific procedures by enabling them to use the same procedures to configure nodes and perform experiments among NETs.

To provide compatibility and complementarity of assistant tools, I propose a layered modular structure. This loosely coupled architecture makes possible the easy coordination among different assistant tools. The tools in this model exchange data with other tools using a simple XML format or text format.

With the aim of emulating a realistic Internet topology, I propose a method to pick up and emulate the proper size of an AS-level network from a public dataset. I also propose a method that emulates an actual operative Open Shortest Path First(OSPF) [9, 10] network topology on NETs.

1.3.3 Implementation and release of AnyBed toolset

I also document the implementation of the proposed architecture as an AnyBed toolset, which has been released as open-source software. This release has attracted several regular users of AnyBed, whose past experiments can be seen in summary form in [11]. For these users, workshops that included an AnyBed session were held in 2008 and 2009 at which they also implemented tools such as XENebula [12], and XBurner [13] using a number of AnyBed components.

Tools for realistic topology design

I designed and implemented several tools that can emulate various parts of the Internet. To emulate inter-AS BGP topology, I implement the caida-topology-filter tool as a part of the AnyBed toolset. Using this tool, users can emulate the proper size of an inter-AS topology based on the actual AS-level topology of the Internet. For intra-AS OSPF topology, I implement ONTES, which is an OSPF Network Topology Emulation System. ONTES infers router configurations from the OSPF Link State Database (LSDB) obtained from one actual router on a backbone network, and then emulates OSPF network topology with the inclusion of the OSPF interface cost settings and assigned IP addresses on NETs. Furthermore, AnyBed can emulate cooperative topology that includes inter-AS BGP topology and intra-AS OSPF topology. This function makes possible end-to-end network emulation from an intra-AS network in AS 1 through an inter-AS network between AS 1 and AS 2 to an intra-AS network in AS 2.

Portable configuration tools among NETs

To provide for reusability of network topologies among various NETs, AnyBed must provide for portability of its own tools among these NETs. Each NET has its own experimental environment that consists of different hardware and different operating systems. AnyBed must be able to work in these environments in order to meet the requirement of system portability.

I have considered several mechanisms that could provide portability of tools. In the end, I adopted the mechanisms that are called the AnyBed portable toolset. Regardless of the original operating systems (OSes) installed in the NET nodes, the master server of the toolset provides the nodes with a specified OS image and software via a PXE boot and NFS root. Users can use same toolset while performing experiments on various NETs accessing one master server that packages this toolset into the NETs.

The general usage of this toolset is as follows. First, after connecting the master server to include the toolset in the first NET, a topology is constructed using this kit. Second, the master server is disconnected and brought to the facilities where another NET is located. Next, after reconnecting it to another NET, it becomes possible to quickly rebuild the same topology by regenerating the actual configuration files.

1.3.4 Prove to expedite network emulation experiments

The study's evaluation results show that AnyBed can expedite network emulation experiments. Using AnyBed, the total time taken to construct a complex BGP topology on 150 nodes was 113 s. This result indicates that AnyBed can generate configuration files and can deploy them on a large scale NET in a short enough time period.

1.4 Organization of this Dissertation

The rest of dissertation is organized as follows. In Chapter 2, I analyze the problems faced when performing experiments on NETs. Next, I proposed AnyBed architecture to solve the problems that are analyzed in Chapter 3 . Next, in Chapter 4, I describe the mechanisms of topology reusability and system portability that are necessary to expedite network emulation experiments across NETs. Next, I describe two topology emulation mechanisms based on the AnyBed architecture: Inter Autonomous System Network Topology Emulation in Chapter 5, and OSPF Network Topology Emulation in Chapter 6. Chapter 7 presents a discussion of various significant issues regarding AnyBed. Finally, Chapter 8 concludes the dissertation by describing the contributions offered by the proposed model and posing questions that remain open to exploration.

Chapter 2

Toward Expeditious Experiments on NETs

In this section, I explore experiments conducted in NETs and outline the general steps involved in performing experiments. Then, I consider the problems in these general steps that are obstacles to the achievement of more expeditious experiments.

2.1 Network Experiments for Research and Development of Network Technologies[1]

Before new technologies are deployed to a real world, they need to be evaluated by network experiments in order to avoid any negative effects they may have upon existing systems. For such an evaluation, the developers of these technologies need to perform *experiment*. In this section, I firstly describe general research and development procedures of network technologies. Next, I introduce experimental methods that can be used on the verification part in the procedure. Then, I discuss reproducibility, that is important factor in experiments.

[1]This section is based on my colleague Toshiyuki Miyachi's paper [14].

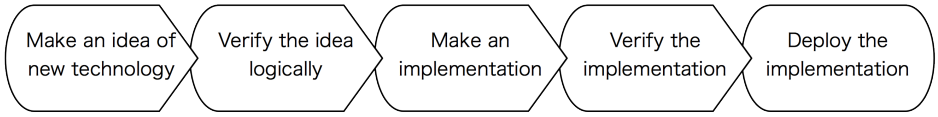

Figure 2.1: Research and Development Procedures of Network Technologies

2.1.1 Research and Development Procedures of Network Technologies

The purpose of a given experiment will be different for different developers. Nonetheless, there are procedures common to them when I focus on the research and development of network technologies. I present the procedure that is considered here in figure 2.1. First, a researcher or developer generates an idea for a new technology. Next, he verifies the idea logically in terms of its effectiveness, which I refer to as *logical verification*. After this, he implements the idea concretely, which means creating software that can be run on suitable hardware. After this, he verifies whether or not the implementation worked as he intended, a step I refer to as *practical verification*. Presuming that it works, the final step is the deployment of the implementation in an actual working environment.

2.1.2 Experimental Methods

The purpose of the step of logical verification is to verify the proper function of the idea in a hypothetical environment before implementing it in an actual one. The effectiveness of his proposed algorithms and principles, for example, needs to be verified. As for the practical verification, its purpose is to verify that the implementation functions well in an environment that is more practical than the abstract realm of logical verification. The developer should confirm that the implementation works properly as he intended, and should also compare how it functioned in this verification with how it functioned in the previous logical verification. Several experimental methods are used for these verifications, depending on their different purposes. Figure 2.2 presents a classification of these experimental methods.

One method is *simulation*, in which an experimenter makes a model of a target technology and its surrounding environment and then evaluates it. The simulation that is normally used for logical verification is sub-divided into *numerical analysis*

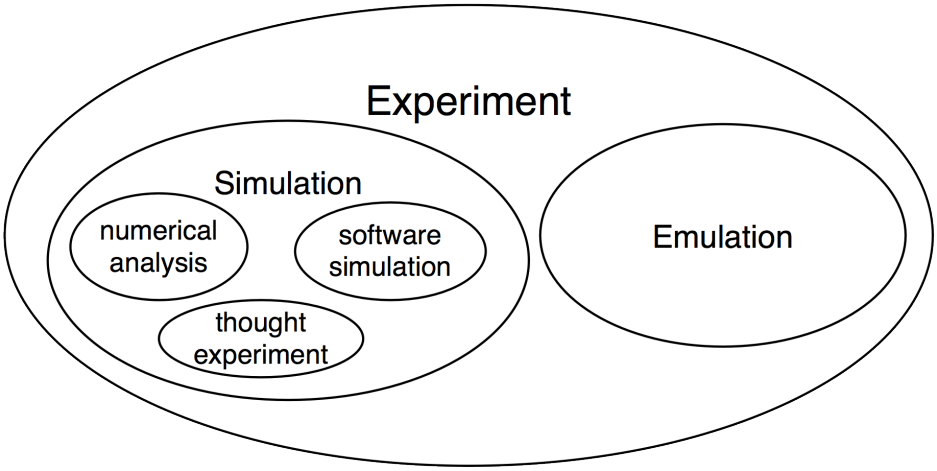

Figure 2.2: Classification of experimental methods

and *software simulation*. The numerical analysis uses numerical calculation based on modeled formulas for its evaluation. In the software simulation, users use the software to implement the models of the target technology. There are a number of infrastructure software products that support general software simulations, the typical ones being ns-2 [6] and SSFnet [15].

Another method is *emulation*, in which an experimenter imitates the surrounding environment of the target technology using alternative hardware or software, and evaluates the target technology by running the software. This method is commonly used for the step of practical verification. In contrast to simulation, in emulation the target technology does not need to be changed because the surroundings simulate the interface to the target technology. This is the advantage of emulation, which makes it possible to reduce the workload involved in changing the implementation for a later deployment.

In network research field, this emulation method is commonly used with Network Emulation Testbed (NET) by many researchers to perform their experiments. A NET is a customized PC-based cluster that enables users to perform a network experiment by running their implementations on its nodes. Some typical examples of experiments performed on NETs are deployment tests, verifications, and performance evaluations of software. In this dissertation, the focus is upon network experiments on NETs.

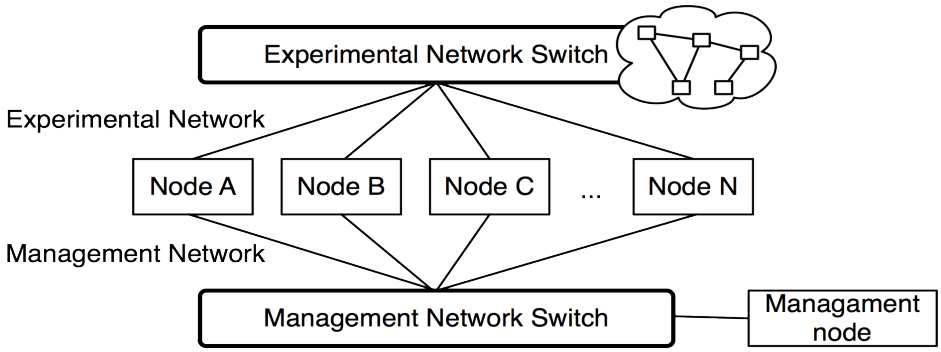

Figure 2.3: Physical topology of NETs

2.1.3 Reproducibility and Repeatability of Experiments

In scientific methodology, reproducibility of experiment is important to repeat the experiment by the other experimenters. Reproducibility means that other experimenter can get the same result when he performs an experiment in the same conditions. Repeatability is also important factor in experiment. Repeatability means that the same experimenter can always get the same result on the same experiment. In case of network simulation and emulation, reproducibility and repeatability is higher than experiments held in other research field because there are no disturbance from outside. However, non-systematic and experimenter-specific procedures can lower reproducibility and repeatability.

2.2 Experiments in a Network Emulation Testbed

This section offers details regarding experiments in a NET. It begins with a description of the general structure of NETs, and then discusses the processes common to network emulation experiments.

2.2.1 Structure of Network Emulation Testbed

Here, I describe the structure of a NET. A NET is a customized PC-based cluster that enables users to perform network experiments by running their implementations on its nodes. The typical physical topology of NETs is shown in Figure 2.3. Each

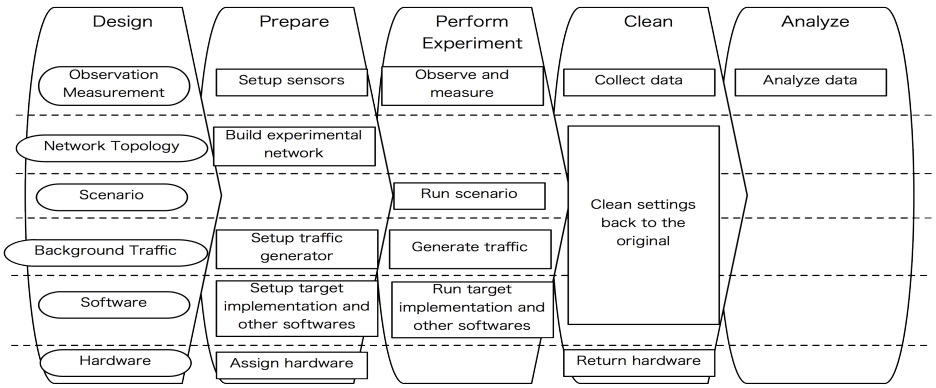

Figure 2.4: Process of network emulation experiments

node on the NET is connected to two network switches: the experimental network switch and the management network switch. The former switch creates layer 2 or layer 3 network topology for the user's experiment using technologies such as Virtual Local Area Network (VLAN) and Asynchronous Transfer Mode (ATM). The term "Network Emulation" denotes an experimental network that is emulated by the switch and nodes. The latter switch creates a management network for controlling the nodes and transferring data related to the experiment. This network also connects to a management node that controls other nodes and collects data.

Each node in a NET acts not only as a router that emulates an experimental network but also as a server that runs a user's implementation. Users perform their experiments with these nodes and the emulated network.

2.2.2 Process of Network Emulation Experiments

From my numerous experiences [11] to support experiments in several NETs, I have analyzed the processes involved in network emulation experiments in NETs. I divide the entire process into five stages whose requisite elements are considered for each stage. Figure 2.4 shows the entire process. The five stages involve the following actions: design, preparation, performance, experimentation, cleaning, and analysis. The requisite elements of these stages have been classified as follows: observation/measurement, network topology, scenario, background traffic, software, and hardware. I describe the details of these stages and the elements of each stage in the following subsections.

Elements for experiments

Although the purpose of an experiment varies with each experimenter, there are elements that are common to most experiments, namely: observation/measurement, network topology, scenario, background traffic, software, and hardware.

Observation and measurement are important elements of experiments. To verify or evaluate a target implementation, observation and measurement are essential processes.

Network topology and background traffic are also commonly used elements. However, the complexity of the topology employed and the type of background traffic will differ, depending on the purpose of a given experiment. For example, a flat network and no background traffic would be suitable for a simple scalability experiment with server software. On the other hand, for the evaluation of a modified transport protocol, a dumbbell topology and heavy background traffic would be required.

Software is composed of a target implementation and other surrounding software programs, which could include OSes, libraries to run the target implementation, and software for observation/measurement, for example. These software programs interact with each other according to a specified scenario.

All these elements are operated with hardware on a NET that includes servers, network switches, and cables.

Design stage

In this stage, the experimenter designs the overall plan of his experiment based on its purpose. To do this, he needs to consider a network topology, a scenario, background traffic suitable for the experiment, a method of observation/measurement, and the software that is required. He must also consider the number of hardware elements and their specifications.

Prepararation stage

From that design, he prepares for the experiment on a NET. This stage involves setting up the sensors for the planned observation/measurement, building an experimental network, setting up a traffic generator, and setting up the target implementation and other software. In large scale NETs, he must secure the assignment of the needed hardware before all these setups.

Performing experiment stage

After finishing the preparation, he performs his experiments by running his scenario, which specifies the entire time sequence of the experiment. For example: first run the specified software, generate traffic 10 s later, run another software 20 s after that, etc.

Cleaning stage

Next, he needs to clean all the settings, software and data stored in the assigned nodes. At the same time, he collects the data and logs measured by the sensors. In the case of large scale NETs, he must return the cleaned hardware.

Analysis stage

Finally, he analyzes the collected data. By processing or visualizing the data, he confirms the results of his experiment.

2.2.3 Problems through the whole process

I identified several problems in the course of the entire process. In the design stage, it is difficult for an experimenter to design the proper experiment for his purpose. In the preparation stage, the performance of experiment stage, and the cleaning stage, the time-consuming workload is burdensome. In the analysis stage, it is difficult to verify the correctness of the experimental results. The solution of the first and second problem would enable experimenters to perform experiments more efficiently and accurately. About the third problem, realizing reproducible experiment by network emulation would solve the problem, because many researchers can perform the same experiment and can verify the correctness if possible.

In this dissertation, I mainly focus on a part of the first problem and a part of second problem. The problems addressed include the difficulty of a time-consuming workload in the building of a large network topology and the difficulty in designing a proper network topology for experiments rapidly enough. The third problem is discussed in the discussion chapter.

2.3 Related work

In this section, I briefly describe various scale NETs and then discuss the assistant tools used on NETs. Next, I introduce technologies used on NETs.

2.3.1 Various scale NETs

There are various scale NETs all around us, the smallest of which is probably the PC a user uses for development. Using recent virtualization technology, a single modern PC can run a number of virtualized OSes that are connected to virtual networks. This environment enables the user to perform simple verification processes, such as checking the performance of his implementation.

A typical mid-sized NET is a PC cluster in a user's laboratory. Due to the decline in the cost of PC servers, it is now common for each laboratory to have a PC cluster comprised of dozens of servers. This cluster is normally used by the members of the laboratory for various purposes such as computation, software development, and the network server function.

In the broader world, large scale NETs that are composed of hundreds of servers are available. I present a brief overview of several of such large scale NETS, and then consider the general steps that have been extracted from complete experiments that were conducted on centralized large scale NETs.

StarBED

StarBED is a centralized large scale NET for practical network system-related experiments, which is located at the NICT Hokuriku Research Center [3]. StarBED is composed of 1070 PC nodes, all of which are commodity 1U rack-mountable servers. Each node is connected to the other nodes via the Ethernet, and the network topology of the nodes is capable of flexibly configuring them using VLAN. Apart from this connection, each node has another NIC connected to a management network. This network is used for network booting, controlling nodes, log transfer, etc. Each node can run over 10 virtual machines, which enables users to create a large scale emulated network environment in which over 10000 nodes are available.

Emulab

Emulab is another large scale NET and is also the name of its toolset. Emulab, which is located at the University of Utah, has 374 PCs, 40 wide-area nodes, 58 Wi-Fi nodes, and a front end to PlanetLab. Researchers use Emulab not only for simple network emulation experiments but also for live-internet experiments using its distribution feature, wireless experiments using its 802.11/software radio, sensor network experiments, and network simulations.

Recently, NETs based on the Emulab toolset have been established in several countries. A representative example of this is DETER.

DETER

DETER [4] is the most well-known large scale NET based on the Emulab toolset. DETER allows security researchers to replicate threats of interest in a secure environment and to develop, deploy, and evaluate potential solutions. The testbed has a variety of hardware devices and supports many popular operating systems. Researchers obtain exclusive use of a portion of a testbed that is configured into a user-specified topology and shielded from the outside world by a firewall. DETER's hardware infrastructure was enhanced by a collection of software tools for traffic generation, statistics collection, analysis, and visualization that were developed in its sister project EMIST [16].

PlanetLab

PlanetLab [5] is a distributed, live testbed that is accessed over the Internet. PlanetLab has over 1000 nodes that are distributed at 494 sites world-wide. Most of the nodes are hosted by research institutions, although some are located in co-location and routing centers

Due to its wide distribution, PlanetLab is commonly used as an overlay network testbed and an experimental deployment platform. The advantage PlanetLab offers researchers is that they can experiment with new services under real-world conditions and on a large scale. The example services outlined above all benefit from being widely distributed over the Internet for three reasons: (1) there are multiple vantage points from which applications can observe and react to the network's behavior, (2) they are in close proximity to many data sources and data sinks, and (3) they are distributed across multiple administrative boundaries.

2.3.2 Assistant tools

StarBED has produced its own assistant toolset called SpringOS [1]. SpringOS has several functions such as resource management, OS and software distribution to each node, VLAN configuration, and the driving of nodes according to the user's scenario.

DETER has a Security Experimentation EnviRonment (SEER) [17,18] that enables security researchers to plan, create, and iterate through a wide range of experimental scenarios with relative ease. SEER integrates various tools for configuring and executing experiments and provides a user-friendly interface on which experimenters can access its tools. Because SEER aims to support a wide range of experimentation requirements, many researchers prefer to interact with DETER through it, which fosters collaboration within the security research community.

2.3.3 Technologies commonly used on NETs

There are several technologies that are commonly used for realistic experiments on NETs: topology generators, link simulators, and traffic generators.

Although the actual network topology of the Internet is composed of a large number of routers, servers, and clients, there are only hundreds or thousands of nodes in even a large NET. To decrease the gap between the actual Internet and a NET, modeling the Internet to reduce the number of nodes is important for experiments on a NET. Many previous research initiatives such as topology generators [19–23] tried to model the topology of the Internet and to generate a small part of it.

The term "link characteristics" indicates bandwidth, delay, jitter, and the rate of link loss among nodes in the Internet. These characteristics can be emulated in a NET by link simulators of OSes such as netem [24], NISTnet [25], and Dummynet [26].

Background traffic is also an important element, especially in a protocol experiment. Several methods to generate background traffic have been proposed. The simplest method is the use of bandwidth benchmark software such as iperf [27] or netperf [28] on nodes in a NET. A more complex method involves generating traffic based on actual traffic, which can be generated by Harpoon [29] and a number of commercial router testers.

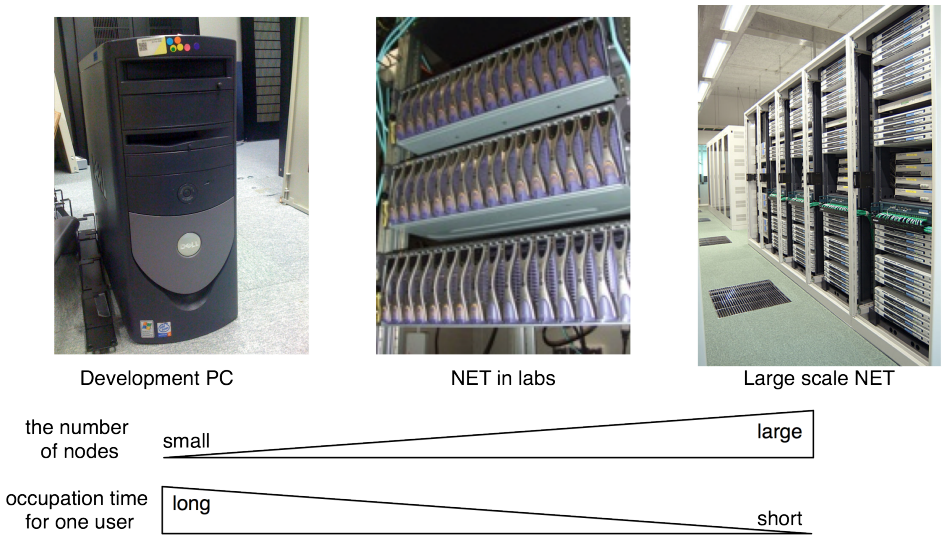

Figure 2.5: Trade-off between scale of NETs and occupational time for one user

2.4 Trade-off between scale of NETs and occupational time for one user

Figure 2.5 shows the trade-off between the scale of NETs and the time they are occupied by a single user. The larger the number of nodes a user needs, the shorter the time that the NET must be occupied.

On a PC used for development, an experimenter can use all the resources on the machine whenever he needs. By contrast, on a large scale NET he can use only part of the servers for a certain period on a reservation basis. These large scale NETs are designed as a shared facility that is simultaneously used by many users. A user generally has to offer some form of compensation, such as a usage fee or an offer of equipment, depending on the time he occupies the NET and the number of nodes used. For these reasons, it is difficult for one user to occupy the majority of the nodes over a long period of time. In any case, if he needs to perform experiments that require many nodes, a user has no alternative but to use a large scale NET.

Due to the trade-off involved, users generally choose a NET of a suitable scale when he needs to perform his experiment.

2.5 General Steps to Perform Experiments on Centralized NETs

What follows is an analysis of the general steps involved in performing experiments on NETs of several different scales. These steps are derived from my experience with laboratory colleagues who performed experiments in several NETs. The steps are described below.

First, the user designs the logical network topology for his experiments. The user also assigns roles for each node; some nodes are used to run programs and others are used to collect the experimental results.

Second, the user assigns physical resources to the designed network topology. For example, an experimental node is assigned to a physical or virtual node, or an experimental interface is mapped to a physical or virtual interface. The user assigns IP addresses to the network interfaces of the experimental nodes. The user assigns VLANs and network addresses to each subnet on the logical network topology. In these sequences, the user must assign appropriate resources along with the physical network topology of a NET, that is, the wiring, the number of physical network interfaces, the bandwidth of these interfaces, and the performance of the CPU.

Third, the user builds the network by setting up physical nodes and layer 2 switches. The user configures network interfaces, routing, and name resolution for each node. The user configures VLANs on layer 2 switches. In addition, the user injects the programs used for the experiment to each node and sets up these programs.

Fourth, the user conducts the experiment, running programs on each node in the specified sequence. Finally, the user collects the results of the experiment and then restores the nodes and switches. The results contain the output of the programs and the state of each node. The results are saved onto the hard disks of each node or transferred to a remote node.

There are various NETs in the world such as desktop PCs, a PC cluster in a laboratory, and large scale centralized NETs. In various testbeds, the user selects a testbed that is suited to the scale of his experiments. For example, during prototype implementation, a desktop PC is sufficient for purposes of testing. However, if he needs a more complex emulated network such as emulated AS-level topology, a desktop PCs will be inadequate, so he would need to move to a larger scale testbed.

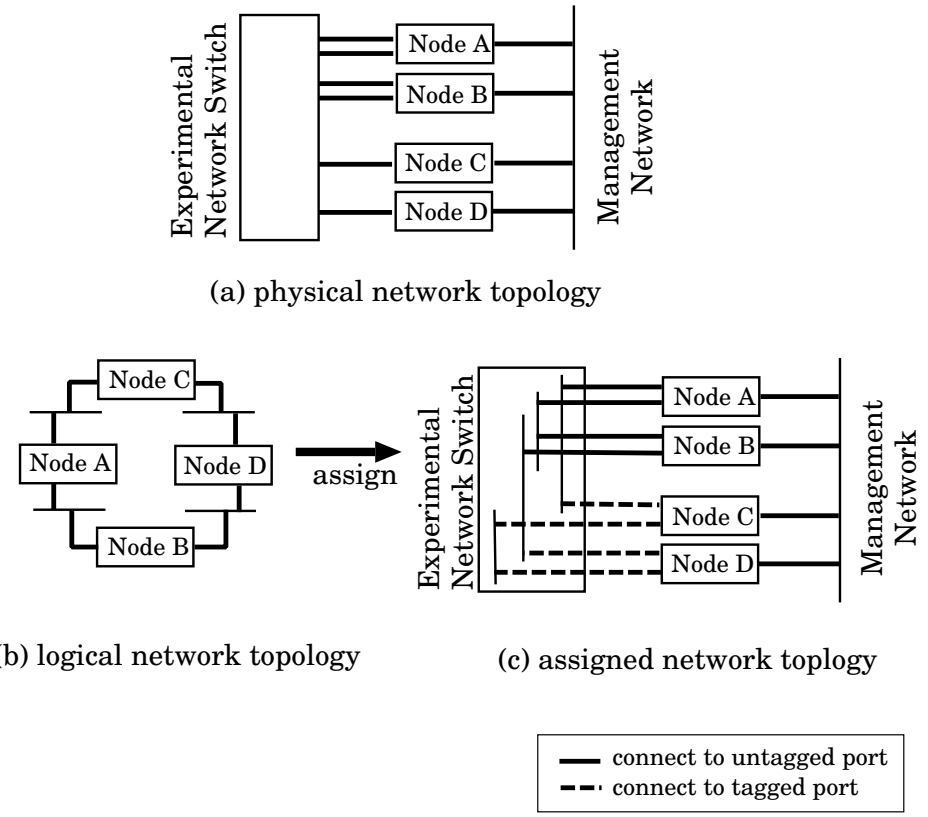

Figure 2.6: Topology assignment

2.5.1 General Steps to Build Experimental Networks

In the steps described above, the work of building an experimental network empirically consumes a proportionately long time in the overall experiment. In addition, as the number of nodes increases, the configuration workload increases as well. For example, when I and my colleagues performed an experiment regarding IP Traceback [30] on StarBED, it took me 4 days out of a one-week (5 working days) reservation to build a logical network topology on 64 nodes and 2 switches of StarBED, on which I emulated the BGP topology of top 50 Autonomous Systems (AS) by using one BGP node as one AS.

In order to expedite the experiments, I focused on reducing the overhead involved in building experimental networks on an NET. I divided the building of experimental networks on an NET into the five steps described below.

1. Design
   The user designs a logical network topology for an experiment (Figure 2.6-(b)). The user considers not only a layer 3 network topology but also a layer 2 network topology.

2. Assignment
   The user assigns the resources of a NET to each experimental node according to the designed logical network topology. The resources to be assigned include PCs, Ethernet Links, VLANs, and IP addresses (Figure 2.6-(a)).

3. Configuration
   The user writes all the configuration files for each application on each node to emulate the designed logical network topology.

4. Injection
   The user injects these configurations to all nodes in actual hardware or software (Figure 2.6-(c)).

5. Check
   The user checks whether the designed logical network topology is correctly built on the NET.

From my experiences, the most time-consuming of the above five steps in network experiments are those of assignment and configuration. The larger the scale of a NET, the longer is the time spent on the assignment step. The more complex a designed network topology, the more items and parameters there are that need to be configured, and the more items or parameters, the more time tends to be spent.

2.6 Problem Analysis

Due to the trade-off and the general steps described in Sections 2.5 and 2.5, a user must move among NETs of several scales many times, depending on the number of nodes required by his experiment. Every experiment has its own time-consuming routine processes such as preparation. I focus upon preparation because if this process can be shortened, the experiment can become more efficient. To achieve this goal, I addressed the following problems in regard to preparation.

No reusability of network topology configuration

I assume that the NET-dependence of network topology configuration can increase the workload involved in reconfiguration. It is difficult to reuse configurations on a different NET because the configurations of a network topology often include resource information that is dependent on specific software, hardware, and/or cable wiring on a NET.

Another reason for the lack of the reusability is that most NET assistant tools are not designed for generating configuration files for a large network topology with several routing daemons. Users must generate their own configuration in addition to the configuration files generated from assistant tools. These user-generated configurations often include the dependent information that was mention above.

Differences in assistant tools and experimental procedures on each NET

Users normally use a different assistant tool on each NET. However, if users could use the same assistant tool and procedures on various NETs, this could reduce the time it takes users to learn NET-specific tools and procedures.

Lack of compatibility and complementarity of assistant tools

Currently, performing an experiment with multiple assistant tools at the same time is a troublesome process. The origin of this problem is the lack of modularity of the assistant tools.

Difficulties in emulating realistic Internet topology

The design of topology for experiments is a time-consuming and difficult process for NET users. Here, I explore a method that can automatically emulate realistic Internet topologies to evaluate Internet applications. These topologies contain inter-AS topologies only, intra-AS topologies only, and mixed topologies that combine both kinds.

Chapter 3

Proposal of AnyBed Architecture[1]

This chapter presents the whole design of AnyBed, which is my proposed architecture for expediting experiments. Based on the problems that I pointed out in Section 2, I firstly show the requirements of AnyBed. Then, I propose the layered modular architecture of AnyBed. We also evaluate the implementation of AnyBed, and confirm that AnyBed can reduce user's configuration workload with portability among NETs.

3.1 Design of AnyBed Architecture

In this section, I describe the design of AnyBed. First, I describe the requirements of AnyBed. Next, I describe the details of the design.

3.1.1 Requirements

According to the problem analysis described in previous chapter, I consider requirements of AnyBed. The design goals of AnyBed are as follows:

- Reusability
  The reusability of logical network topologies beyond differences of physical features among NETs eases a user to quickly start experiments with the same logi-

[1]This chapter is based on my paper "Expediting experiments across testbeds with AnyBed: a testbed-independent topology configuration tool" appeared in Proceedings of 2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom), March, 2006.

cal network topology even if the user tries to perform the experiment in different NETs.

- Portability
  To achieve the reusability of logical network topologies, AnyBed must have its own system portability among NETs.

- Affinity
  Cooperating between AnyBed and other assistant tools of each NET can expedite building experimental networks more quickly. AnyBed should have affinities with each assistant tool on each NET.

- Scalability
  To perform experiments on various scale NETs, AnyBed must work well on a large scale NET as well as small one. Also AnyBed must map a large logical network topology to the large scale NET in short time.

In order to achieve these goals, I divide information for an experiment into two parts. The detail is described in the following section.

3.1.2 Required information to build network topology

In section 2.2.2, I describe several elements required for experiments. Here, when focusing on building network topology, I extract minimum required information from these elements.

Firstly, I consider minimum information to build layer 1 and layer 2 topology. Figure 3.1 shows a physical network on NET. From this figure, required information is desired layer 2 topology, wiring link information among switches and NICs of nodes, VLAN information, node information, and switch information. The node information and the switch information includes existing nodes/switches and its capability information such as VLAN support. If a user obtain these information, he can build layer 1 and layer 2 topology by configuring switch.

Next, I consider information to build layer 3 network topology with figure 3.2. Required information is desired layer 3 network topology. This topology includes IP address information, OSPF information such as link costs, and BGP information.

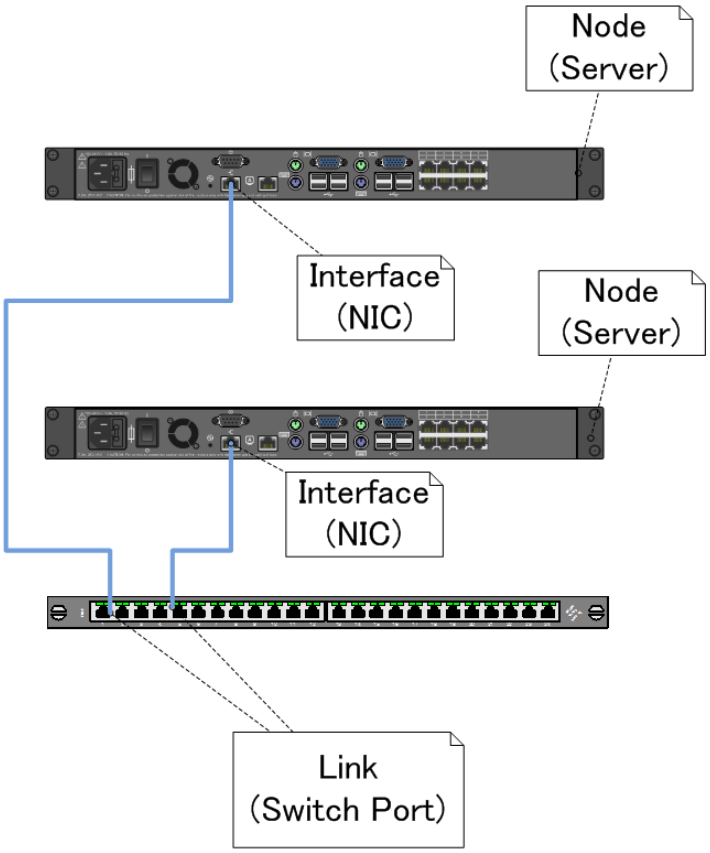

Figure 3.1: A physical network on NET

Consequently, minimum information to build network topology is as follows: desired layer 2 topology, wiring information among switches and nodes, VLAN information, switch information, node information, and desired layer 3 topology.

These information can be divided into two parts: NET-specific information and experiment-specific information. The NET-specific information includes wiring link information, VLAN information, switch information, and node information. The experiment-specific information is desired layer 2 topology and desired layer 3 topology.

Using these divided information, I propose layered modular architecture that is described in the following section.

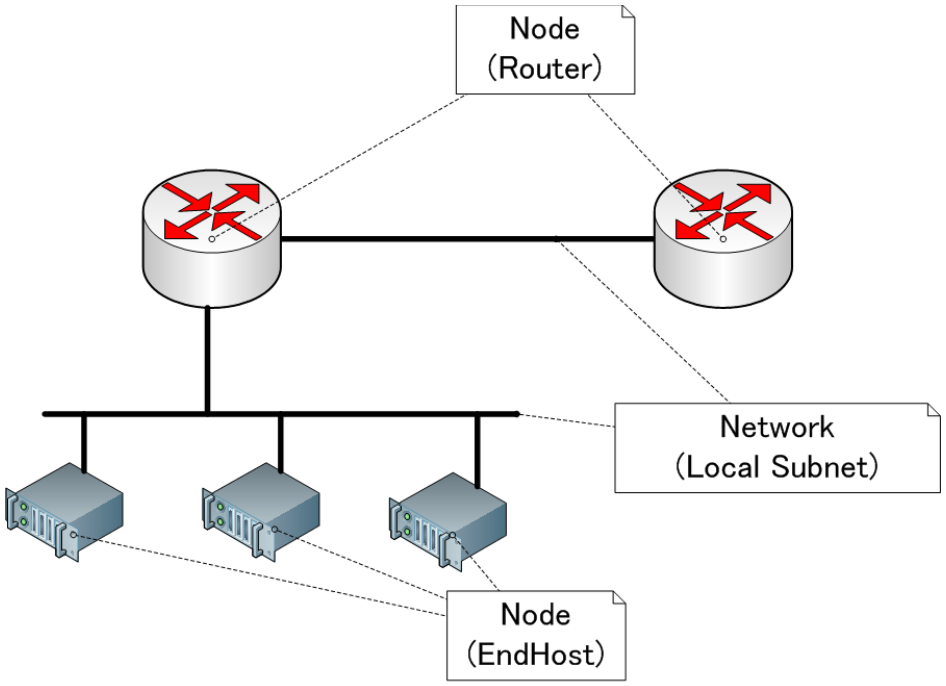

Figure 3.2: A logical network on NET

3.1.3 Layered modular architecture

According to the requirements mentioned in Section 3.1.1, I design the layered modular architecture for AnyBed shown as Figure 3.3. AnyBed consists of three layers: the design layer, the assignment layer, and the injection layer.

In the design layer, a user designs a logical network topology and creates a logical network file according to the designed logical network topology. Similarly, each NET prepares a physical network file along with its physical network topology, that is, its facilities and the wiring among facilities. The assignment layer assigns resources such as PCs, Ethernet links, VLANs, and IP addresses. After then, assignment layer generates actual configuration files for each node and each switch. The injection layer injects actual configuration files to each node, and switch. As this layer depends on OSes and hardware specifications on each NET, I design the layer to cooperate with existing NET-specific toolsets.

Because of the layered module architecture, each layer is easily pluggable. It is easy

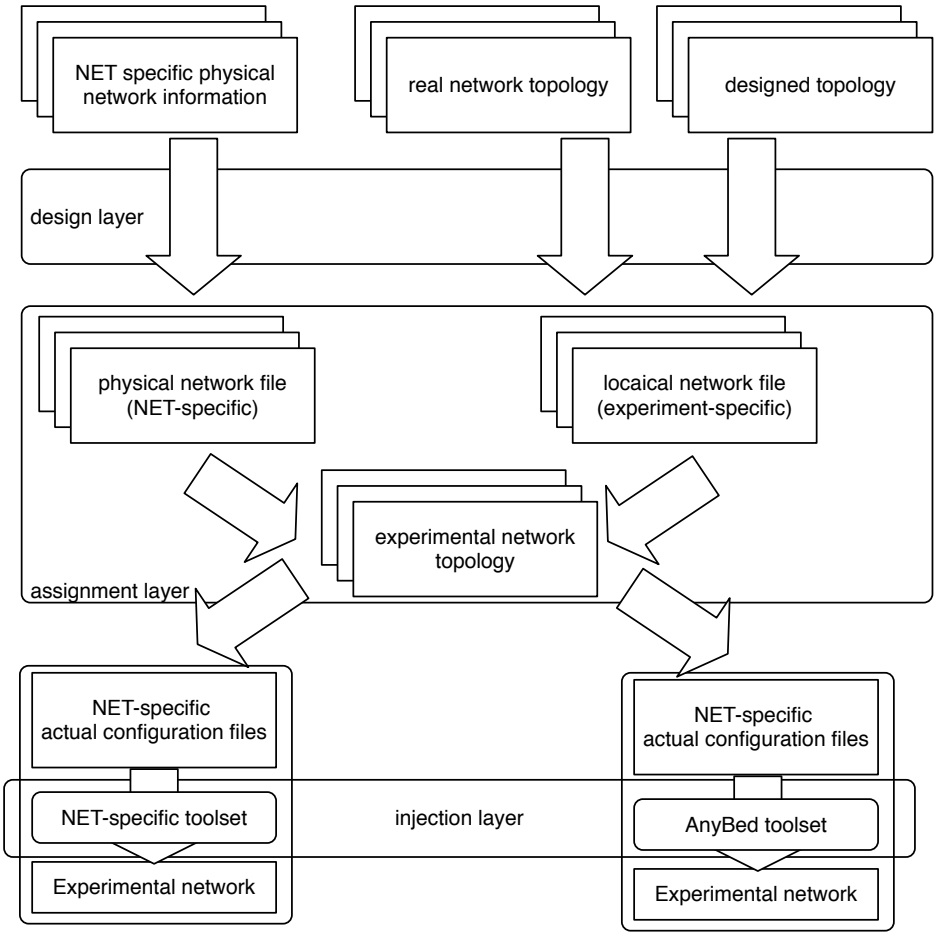

Figure 3.3: AnyBed design

to replace a component on each layer to another and to provide various components on each layer for some specific purposes.

3.1.4 Logical and physical network topology

In AnyBed, a network topology for an experiment is divided into two topologies described in XML format: logical network topology and physical network topology. A logical network topology contains the information about layer 2 and layer 3 topology of an experimental network. Since elements and attributes of a logical network topology present only the connections among logical nodes, it does not depend on a

specific NET environment. On the other hand, a physical network topology shows the information about physical nodes, network bandwidth and the wiring among physical nodes. Hence, the physical network topology of each NET depends on the facilities of each NET.

By using XML for the syntax of each network topology file, network topologies are not only human readable but also easily parsed by computers. Also, the consistency between a logical network topology and a physical network topology can be verified in the XML parser. As for a physical resource description format in XML, GENI project is currently standardizing GENI RSpec [31], which is based on the resource description of Emulab. However, the target of RSpec is only resource description, not topology description.

3.2 Implementation

In this section, I describe the first AnyBed's implementation. The first AnyBed implementation was developed on FreeBSD 4.8R and Ruby 1.8.1. In this first implementation, cluster nodes needed to work with FreeBSD 4.x. Supported layer2 switches are DELL PowerConnect switches and Extreme Networks Summit switches. The first AnyBed implementation covered IPv4, and hired only zerba OSPF daemon for routing. Components of the first AnyBed implementation are shown in Figure 3.4.

In assignment layer, "*dispatcher*" reads a physical network file and a logical network file, assigns physical elements to logical ones, and makes XML topology file. Then, "*config generator*" reads an XML topology file, and makes actual configuration files.

In injecting layer, "*config injector*" injects actual configuration files to each cluster node and switch.

The details of each component are described below.

3.2.1 Configuration files for Network Topologies

In this section, I explain configuration files for network topologies: physical network file and logical network file.

Physical network file describes wiring and capability of nodes and switches. Currently, only the information of interface can be described.

The example of physical topology file is shown in Figure 3.5. In the first level, a

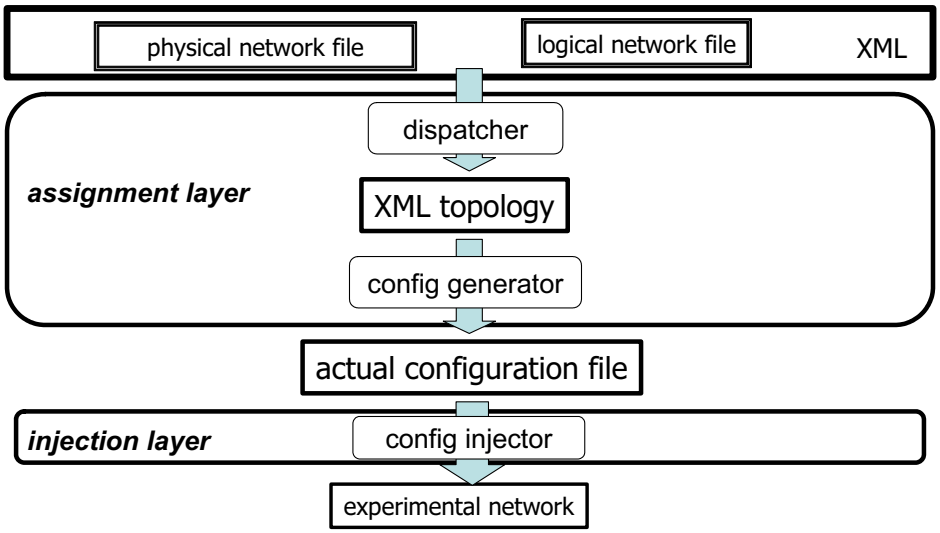

Figure 3.4: AnyBed programs

`<nodes>` presents for node sets. A `<nodes>` has some `<node>` that present physical nodes. A `<node>` has some `<interface>` that present network interfaces on each physical node. A `<interface>` has some `<link>` that present physical links on each interface.

In this example, the node named "mc12" has the management interface "bge0" and has the experimental interfaces named "bge1", "bge2". Each experimental interface connects to the port named "ethernet 1/2" and "ethernet 1/7" in the switch named "mc1-sw2".

Logical network file describes logical network topology that the user desires to build.

The example of logical network file is shown in Figure 3.6. In the first level, a `<nodes>` presents for node sets. A `<nodes>` has some `<node>` that present experimental nodes. A `<node>` has some `<interface>` that present network interfaces on each experimental node. A `<interface>` has some `<network>` that present network on each interface.

The example describes that the "NodeA" has interfaces named "NodeA-Int1" and "NodeA-Int2". And, each interface belongs to "Net1" and "Net2".

```
<nodes>
 <node name="mc12" os="FreeBSD">
  <interface name="bge0" bandwidth="1000"
             dot1q="yes" purpose="management"
             managementip="172.16.1.12">
  </interface>
  <interface name="bge1" bandwidth="1000"
             dot1q="yes" purpose="experiment">
   <link tonode="mc1-sw2" toint="ethernet 1/2"/>
  </interface>
  <interface name="bge2" bandwidth="1000"
             dot1q="yes" purpose="experiment">
   <link tonode="mc1-sw2" toint="ethernet 1/7"/>
  </interface>
 </node>
</nodes>
```

Figure 3.5: Example physical topology

3.2.2 Assignment layer

In the assignment layer, dispatcher reads physical network file and logical network file. Then, dispatcher assigns physical elements to logical ones properly like figure 3.7. The physical elements are a physical node, a network interface in the node, an IP address, a VLAN and bandwidth. Currently, the criteria for properness is only fairness of bandwidth of each physical link.

In logical network file, there is a tree structure: “node” - “interface” - “network”. Similarly, in physical network file, there is a tree structure: “node” - “interface” - “link”. Dispatcher reads these structures, and then make assignment based on the algorithm shown in Figure 1.

The reason I adopt the simple iterative algorithm is that low computational effort is more important than bandwidth optimality. If I seek bandwidth-optimum, I must solve knapsack problem. On the large scale cluster, this cost high computational effort, and make the user wait for a long time.

After dispatcher makes assignment of nodes and interfaces, dispatcher assigns VLAN ID and IP addresses to each interface. These IP addresses are private addresses as 10.0.0.0/8. Next, dispatcher makes XML topology file. Then, config generator translate from XML topology file to actual configuration file. On the current imple-

```
<nodes>
 <node name="NodeA">
  <interface name="NodeA-Int1">
   <network name="Net1"/>
  </interface>
  <interface name="NodeA-Int2">
   <network name="Net2"/>
  </interface>
 </node>
</nodes>
```

Figure 3.6: Example logical topology

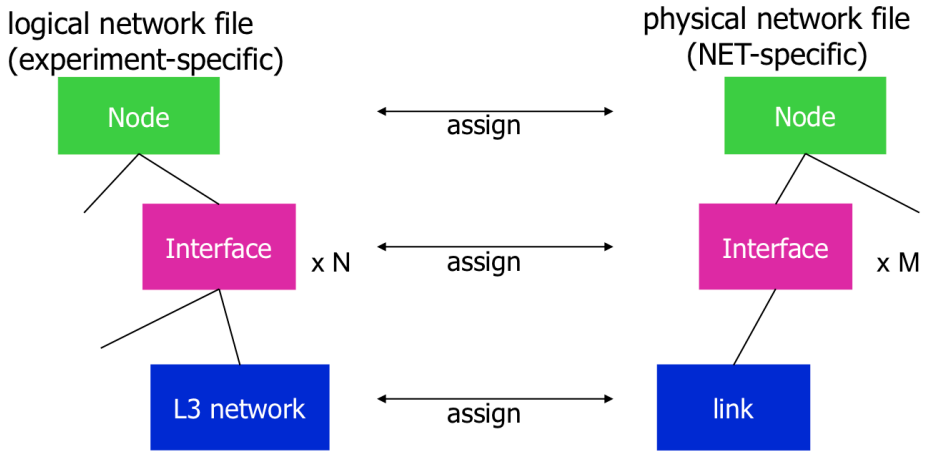

Figure 3.7: Assignment of a physical network file and a logical network file

mentation, dispatcher and config generator are not divided. Dispatcher directly makes actual configuration files.

Actual configuration files are rc.conf, hosts, zebra.conf, ospfd.conf and switch configuration files. Figure 3.8 shows a emulated ospf topology constructed by AnyBed.

3.2.3 Injection layer

In the injection layer, config injector injects actual configuration files to each cluster node and switch, and then inject them. The implementation of config injector would be different for each cluster environment.

Algorithm 1 An algorithm of assignment nodes and interfaces

```
1: procedure Assignment Main Routine
2: sort(LogicalNodes) order by Number_of_Required_Interfaces
3: sort(PhysicalNodes) order by Number_of_Required_Interfaces
4: sort(LogicalNodes.Interfaces) order by BandWidth
5: sort(PhysicalNodes.Interfaces) order by BandWidth
6: for all LogicalNodes do
7:     l ← LogicalNode.Interface
8:     p ← PhisicalNode.Interface
9:     for all l do
10:        p.ResidualBandwidth ← p.TotalBandwidth
11:        MaximumInterface ← maximum(p) order by Bandwidth
12:        MaximumInterface.VlanID = assignNewVlanID(MaximumInterface)
13:        AverageBandwidth ← p.TotalBandwidth / l.TotalBandwidth
14:        p.ResidualBandiwth ← p.CurrentBandwidth - AverageBandwidth
15:        remove MaximumInterface from l
16:    end for
17: end for
```

We use three types of config injector. The first is using the modified version of "mkdiskimage" injected from StarBed [3]. The mkdiskimage is the environment for making memory filesystem image of PXE [32] boot. When a cluster node boots, the node gets two contents: memory filesystem image from TFTP server and tarball from FTP server. The memory filesystem image is mounted to root filesystem, and tarball is extracted there. Then, operating system reads configuration files there. In the tarball, we pack actual configuration files.

The second is the method for injecting actual configuration files over NFS. When a cluster node boots, the node gets actual configuration files from mounted NFS directory.

The third is the program that communicates to switches via TELNET.

3.3 Verification and Evaluation

In this chapter, I describe the verification result and evaluation result of the AnyBed implementation. We conducted the following verifications and evaluations: topology reusability, workload, and scalability to the number of nodes.

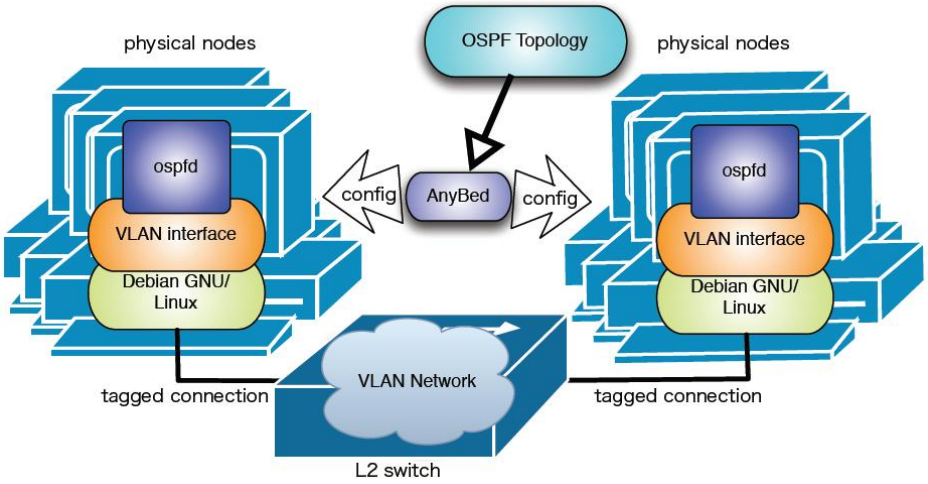

Figure 3.8: An emulated OSPF topology by AnyBed

Table 3.1: Verification environment for reusability and workload (homogeneous cluster)

Node#1 to Node#17	CPU	Intel Pentium3 1.4GHz
	Memory	1024MB
	NIC	Broadcom BCM5703X(1000Mbps) x 2
Layer2 switch		DELL PowerEdge1655MC Switch

3.3.1 Topology reusability and workload

In this section, I verify topology reusability and workload about AnyBed. We prepare two clusters that their hardware equipment is different each other. On these two clusters, I build the same network topology by AnyBed, and I compare number of configuration files, size of configuration files, average time of building network.

Equipments on each cluster are described in Table 3.1 and Table 3.2. The latter cluster has heterogeneous nodes while the former cluster has homogeneous nodes.

The homogeneous cluster comprises of 17 blade servers and a layer 2 switch made by Dell. Physical topology of the cluster is shown in Figure 3.9. We use one server for DHCP, FTP server, and the other for building experimental network. These cluster node are connected with each other by 6 layer2 switches named like mcX-swY in the figure. Because all equipped NICs are PXE-capable, I use PXE boot and mkdiskimage

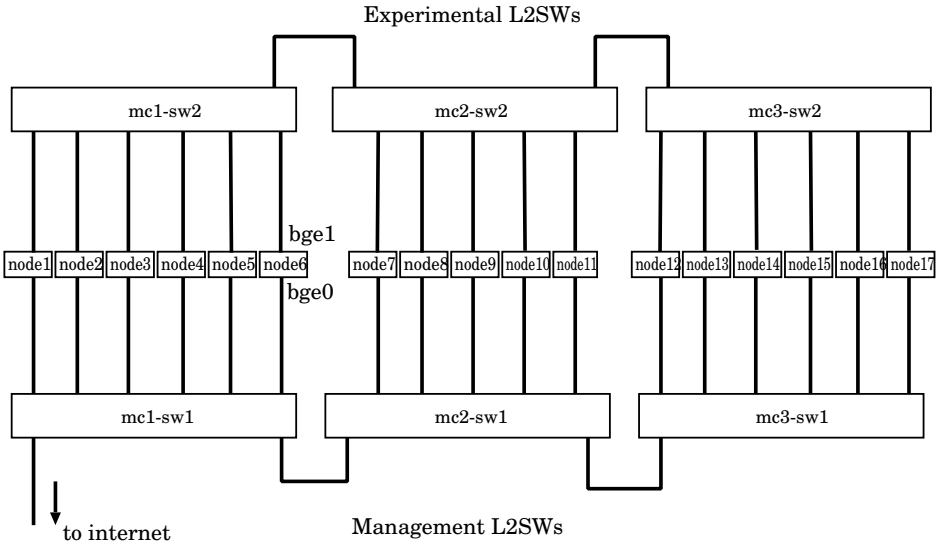

Figure 3.9: Physical topology of my homogeneous cluster

for config injector.

The heterogeneous cluster is composed of 7 nodes. Physical topology of the cluster is shown in Figure 3.10. Although there are 10 nodes in the figure, neta01, Node#1 to node#3 and node#4 to node#7 are different in the type of NIC. Therefore, in two groups, bandwidth and the name of network interface in operating system is different.

In the heterogeneous cluster, I cannot use PXE boot for config injector because all equipped NICs are not PXE-capable. Instead of PXE boot, I use the method using NFS for config injector.

On these 2 clusters, I give the same logical network file that 7 routers are connected with full mesh, then verify whether the same experimental network topologies are built or not. The used topology is shown in Figure 3.11. To investigate network topologies, I login to Zebra OSPF daemon via telnet, and get routing information by executing "show ip ospf route" command. One of the obtained routing information is shown in Figure 3.12. Obtained routing information in both cluster has the same routes and the same routing costs. This result means the same network topology including the same link structure and link costs among routers have been built in both cluster, I consider. Simultaneously, I measure the time until experimental network is built. These sequence is repeated 10 times to calculate the average. In addition, I compare

Table 3.2: Verification environment for reusability and workload (heterogeneous cluster)

Node#1 to Node#3	CPU	Intel Pentium3 450MHz
	Memory	256MB
	NIC	Intel Pro 10/100B/100+(100Mbps)
		3Com 3c905B-TX(100Mbps)
Node#4 to Node#7	CPU	Intel Pentium3 900MHz
	Memory	256MB
	NIC	3Com 3c905B-TX(100Mbps)
		Netgear GA620(1000Mbps)
Layer2 switch		FoundryNetworks EdgeIron4802F
		ExtremeNetworks Summit48
		ExtremeNetworks Summit5i

Table 3.3: Verification result of usability and workload

	Homogeneous	Heterogeneous
Number of configuration file in AnyBed	2	2
Size of configuration file in AnyBed	10671 Byte	6549 Byte
Number of actual configuration files	31	30
Size of actual configuration files	33957 Byte	39191 Byte
Average time of building network	137 sec	135 sec

number of configuration files, size of configuration files on 2 clusters. Verification result is described in Table 3.3.

First, I discuss about size and number of configuration files. We use the logical network file that size is 3377 Byte on both clusters. About the size of physical network file, the size of the file in homogeneous cluster is 7294 Byte, and the size of the file in heterogeneous cluster is 3172. On the other hand, the total size of actual configuration files are 33957 Byte in homogeneous cluster, and 39191 Byte in heterogeneous cluster. About the number of configuration files, the number of the file in homogeneous and heterogeneous cluster is 3, while the number of the actual configuration files in homogeneous cluster is 31, and the number in heterogeneous cluster is 30. Without AnyBed, the user must write the total size and total number of the file by hand or by script.

Second, I discuss the time of building experimental network. The time of building

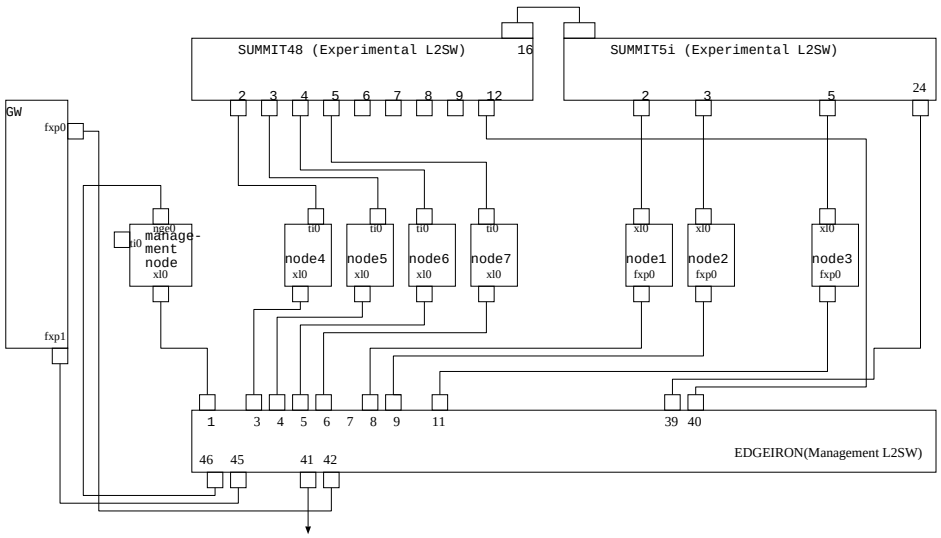

Figure 3.10: Physical topology of my heterogeneous cluster

experimental network is fewer 140 sec on both clusters. We think that 140 sec is short enough compared to build it by hand.

Therefore, workload that the user write configuration files is reduced by using AnyBed.

3.3.2 Scalability to the number of nodes

In this section, I evaluate whether AnyBed can treat large scale network and many nodes. Because I do not have real large scale cluster, I make dispatcher reading logical network file that describes many nodes and networks.

We describe evaluation environment. Because I suppose to use AnyBed in StarBed, I described the information of nodes that is the same as StarBed which the number of nodes is 512.In logical network file, I describes the structure where each node have fullmesh link as much as possible. The fullmesh link mostly increases the size of configuration files and complexity of network topology. On the real network, It is impossible for 512 nodes to have fullmesh link each other because VLAN ID is limited upto 4096. We assume usable VLAN ID is up to 4000 because of reservation of VLANs for management. Therefore, I use the topology that is partly fullmesh and the rest is tree. Up to 88 nodes makes fullmesh link and the rest is tree that the root is fullmesh

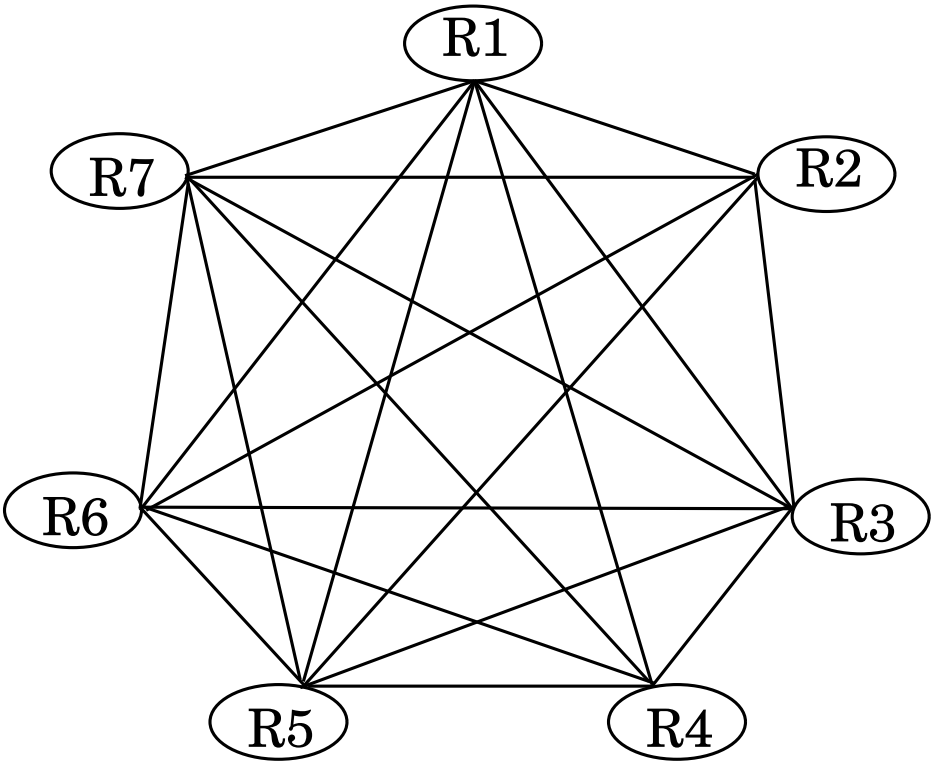

Figure 3.11: Full-mesh layer 3 network topology for verification of reusability

node. Using this topology, I run dispatcher and measure time consumption, memory consumption with changing the number of nodes from 2 to 512 on the environment described in Table 3.4.

The results are shown in Figure 3.13, Figure 3.14, and Figure 3.15. On 512 nodes, time consumption of dispatcher is 579sec, and time consumption of the part of assignment algorithm is 2sec, and memory consumption is 211 MB.

Figure 3.13, Figure 3.14, and Figure 3.15 shows that time consumption and memory consumption increase drastically up to 88 nodes. That is why the number of network increase drastically up to 88 nodes where the topology is fullmesh and increase slowly above 88 nodes where the topology is tree.

```
============ OSPF network routing table ============
N    10.0.0.0/24           [20] area: 0.0.0.0
                           via 10.0.16.2, vlan1
                           via 10.0.18.2, vlan2
N    10.0.1.0/24           [20] area: 0.0.0.0
                           via 10.0.15.2, vlan0
                           via 10.0.20.2, vlan3
N    10.0.2.0/24           [10] area: 0.0.0.0
                           directly attached to vlan4
N    10.0.3.0/24           [20] area: 0.0.0.0
                           via 10.0.16.2, vlan1
                           via 10.0.20.2, vlan3
N    10.0.4.0/24           [20] area: 0.0.0.0
                           via 10.0.15.2, vlan0
                           via 10.0.2.2, vlan4
N    10.0.5.0/24           [10] area: 0.0.0.0
                           directly attached to vlan5
N    10.0.6.0/24           [20] area: 0.0.0.0
                           via 10.0.18.2, vlan2
                           via 10.0.20.2, vlan3
N    10.0.7.0/24           [20] area: 0.0.0.0
                           via 10.0.2.2, vlan4
                           via 10.0.16.2, vlan1
N    10.0.8.0/24           [20] area: 0.0.0.0
                           via 10.0.15.2, vlan0
                           via 10.0.5.2, vlan5
N    10.0.9.0/24           [20] area: 0.0.0.0
                           via 10.0.2.2, vlan4
                           via 10.0.18.2, vlan2
N    10.0.10.0/24          [20] area: 0.0.0.0
                           via 10.0.5.2, vlan5
                           via 10.0.16.2, vlan1
N    10.0.11.0/24          [20] area: 0.0.0.0
                           via 10.0.2.2, vlan4
                           via 10.0.20.2, vlan3
N    10.0.12.0/24          [20] area: 0.0.0.0
                           via 10.0.5.2, vlan5
                           via 10.0.18.2, vlan2
N    10.0.13.0/24          [20] area: 0.0.0.0
                           via 10.0.5.2, vlan5
                           via 10.0.20.2, vlan3
N    10.0.14.0/24          [20] area: 0.0.0.0
                           via 10.0.2.2, vlan4
                           via 10.0.5.2, vlan5
N    10.0.15.0/24          [10] area: 0.0.0.0
                           directly attached to vlan0
N    10.0.16.0/24          [10] area: 0.0.0.0
                           directly attached to vlan1
N    10.0.17.0/24          [20] area: 0.0.0.0
                           via 10.0.15.2, vlan0
                           via 10.0.16.2, vlan1
N    10.0.18.0/24          [10] area: 0.0.0.0
                           directly attached to vlan2
N    10.0.19.0/24          [20] area: 0.0.0.0
                           via 10.0.15.2, vlan0
                           via 10.0.18.2, vlan2
N    10.0.20.0/24          [10] area: 0.0.0.0
                           directly attached to vlan3
```

Figure 3.12: OSPF routing table

Table 3.4: Evaluation environment for scalability

Hardware	
CPU	UltraSPARCIII 900MHz × 24
Memory	64GB
Software	
Operating System	Solaris 8
Programming Language	Ruby 1.8.1
XML parser	REXML 2.4.8

3.3.3 Summary of verification and evaluation

Because of the verification and evaluation result which is described above, AnyBed enables logical topology reusability on different NETs, building experimental network of 16 nodes below 140sec, reducing configuration file size as 1/3, number of configuration files as 1/15.

Therefore, the user can perform experiment by the same testbed environment on various clusters.

3.4 Summary

Through my experiences performing experiments on NETs, I pointed out merits for the case that a assistant system becomes portable across NETs. With considering the portability, I analysed the steps to perform experiment on NETs. In the steps, I particularly forcused on the step to build experimental networks because it is time-consuming in the all steps. Considered with procedures in the step to build experimental networks, I designed AnyBed architecture.

As the first implementation, I developed assignment layer and injection layer of AnyBed. We also verified and evaluated the assignment layer implementation. A result of the verification and evaluation showed that a user can perform experiment by the same testbed environment portably on various NETs.

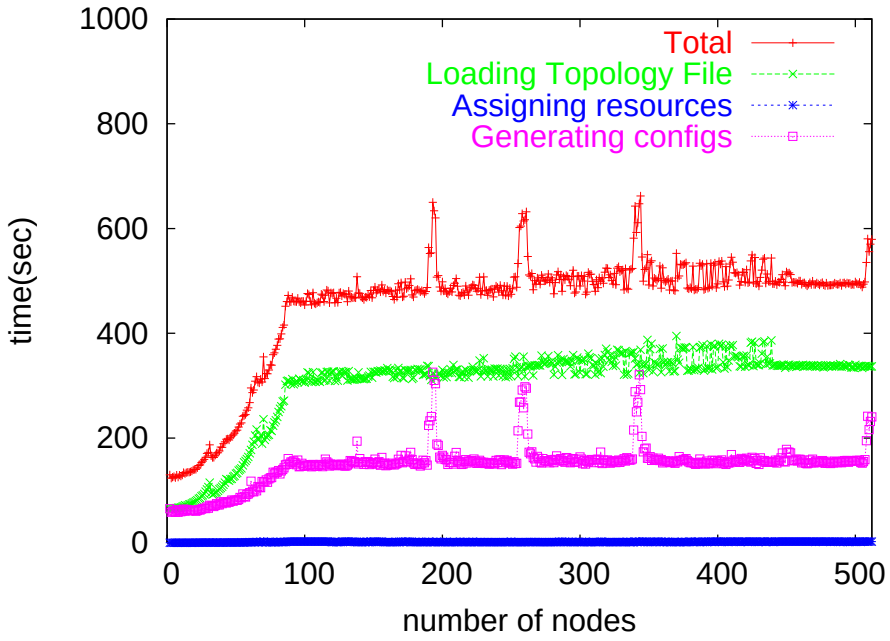

Figure 3.13: Time consumption of dispatcher under different number of nodes

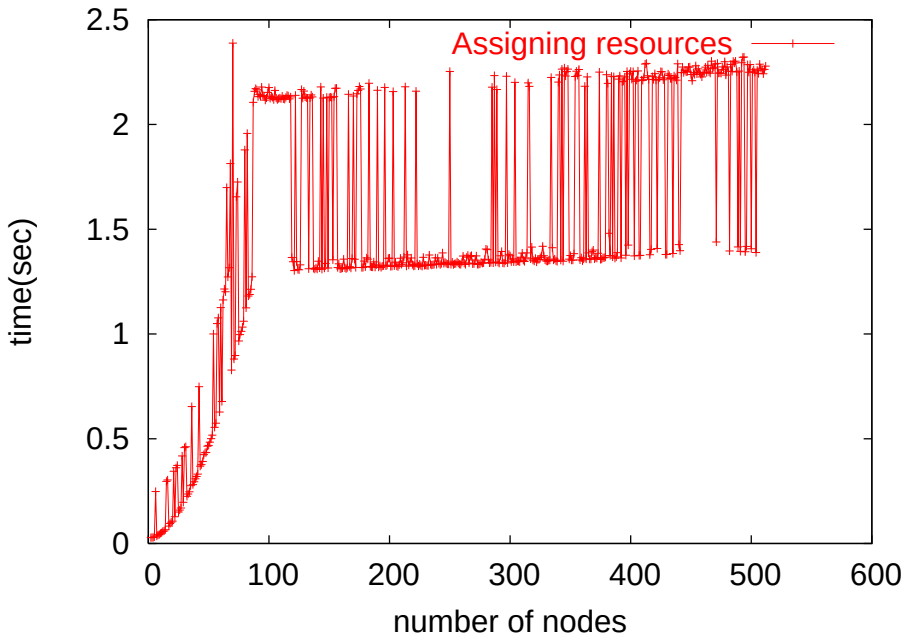

Figure 3.14: Time comsumption of the part of assignment algorithm in dispatcher under different number of nodes

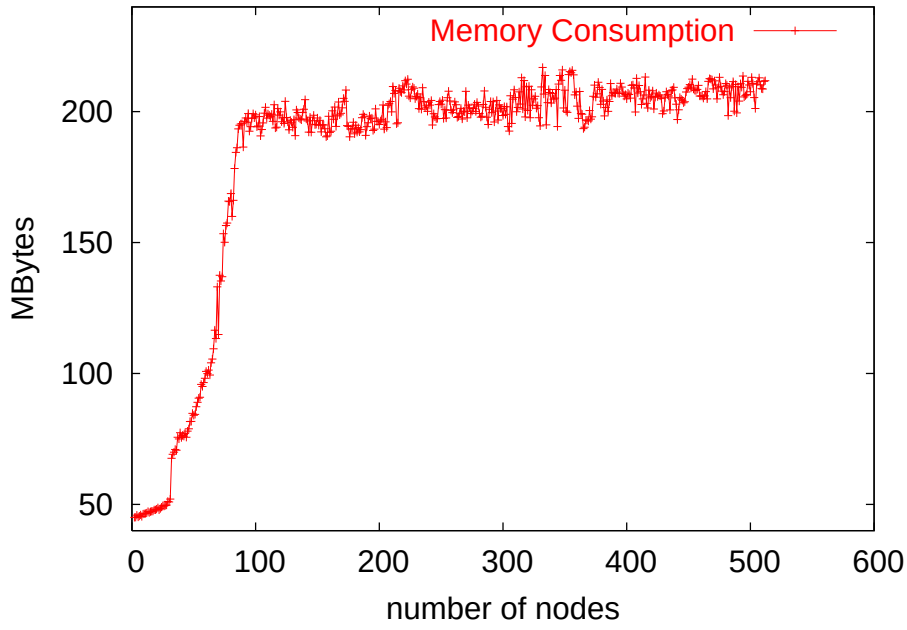

Figure 3.15: Memory consumption of dispatcher under different number of nodes

Chapter 4

Topology Reusability and System Portability[1]

This chapter presents considerations about topology reusability and system portability, which are the core ideas of AnyBed. Firstly, I describe the importance of topology reusability and system portability on NETs. Then, I show the design of AnyBed realizing the reusability and the portability. Finally, I show the results of verification and evaluation.

4.1 Issues of Topology Reusability and System Portability

Due to the trade-off and the general steps described in Section2.5 and 2.5, a user have to move several scale NETs many times depending on the necessary number of nodes about his experiment. On such movements among NETs, configuration files of assistant tools and tools itself are currently customized for specific NETs. This traditional design increses a user's workload such as regenerating configurations to

[1]This chapter is based on a journal "Expediting Experiments across Testbeds with AnyBed: A Testbed-Independent Topology Configuration System and Its Tool Set" in *IEICE Transactions on Information and System*, Vol. E92-D, No. 10, pp. 1877.

I edited this journal to focus on the portability of AnyBed system and experiments across various NETs. I referred the experiences of my colleagues who did several experiments with AnyBed. The experiences of my colleagues was written in [11].

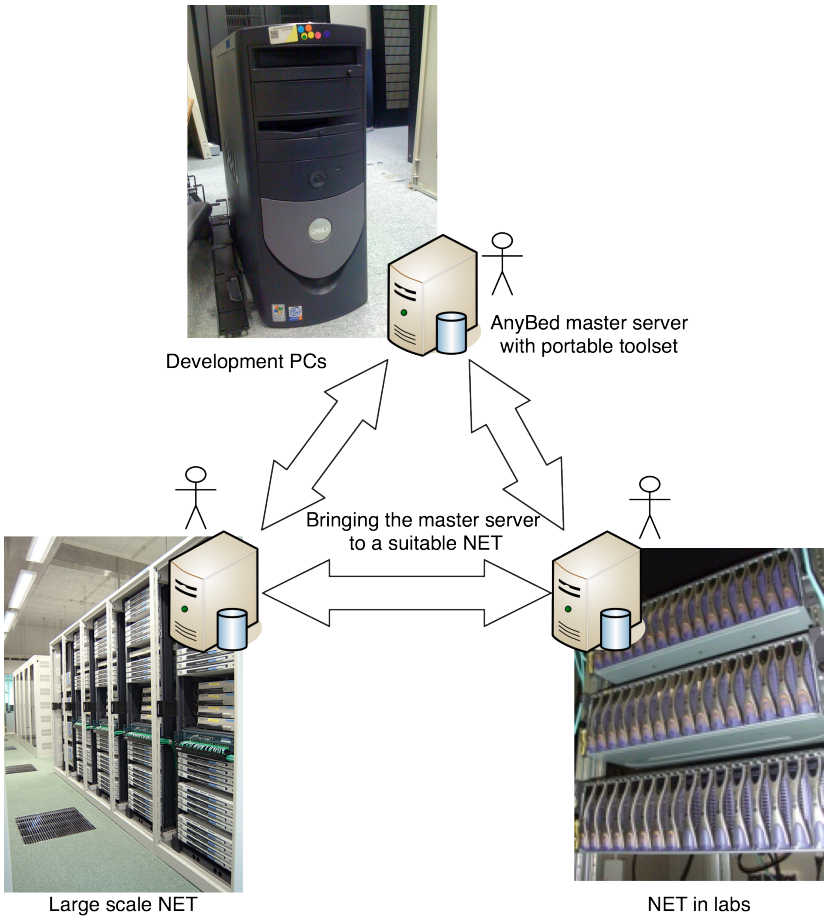

Figure 4.1: A proposed NET use model with AnyBed

move among various scale NETs.

Topology reusability and system portability can reduce this workload. In this chapter, I explore the way to enable topology reusability and system portability.

4.2 A NET use model with AnyBed

In this section, I explain a proposed model for users to use NET with AnyBed in Figure 4.1. This model supposes three behavior of users.

The first behavior is that a user should select suitable scale NET for his experiment. For example, during prototype implementation, it is enough to test it by his desk-side

PCs. However, if he needs more complex emulated network like emulated AS-level topology, he cannot perform experiments by his desk-side PCs. He may move to a larger scale testbed.

The second one is that the user should use the same assistant tools among various NETs. The user need not to change tools or procedures when he moves to other NET.

Ther last one is that the user should reuse configurations about experimental network topology. The user does not have to regenerate the configurations after each move.

To realize this model, we design and implement AnyBed portable toolset and cooperative tools. The details of the tools are described in next section.

4.3 Cooperative tools for AnyBed

In this section, I describe cooperative tools for AnyBed. First, I describe about AnyBed portable toolset that makes AnyBed portable among NETs. Then, I introduce implementations of injection layer tools and implementations of design layer tools.

4.3.1 AnyBed portable toolset

For reusability of logical network topologies among various NETs, AnyBed must have its own system portability among NETs. Each NET has its own experimental environments consisted of different hardwares and different operating systems. AnyBed must work on these environments to meet the requirement of system portability. Additionally, performing experiments among various NETs by the same tools gives users another advantage to omit learning NET's own tools at every NET.

Because of such situations, I designed a toolset called "AnyBed portable toolset" that I can be commonly used in various NETs. This toolset contains the following functions. (1) The user can use the OS which he wants without bothering with whether HDD is equipped or not and which OS was installed on experimental nodes. The tool contains servers of DHCP, TFTP, and NFS to boot clients by PXE(Preboot eXecution Environment) [32]. After nodes booted, they use NFS as its root file system, not depend on its equipped HDD. (2) The user can command each node to perform experiments without using NET's own toolset. This tool named "gsh" is based on "DSH(Distributed Shell)" [33]. I can use same toolset while performing experiment on various NETs by

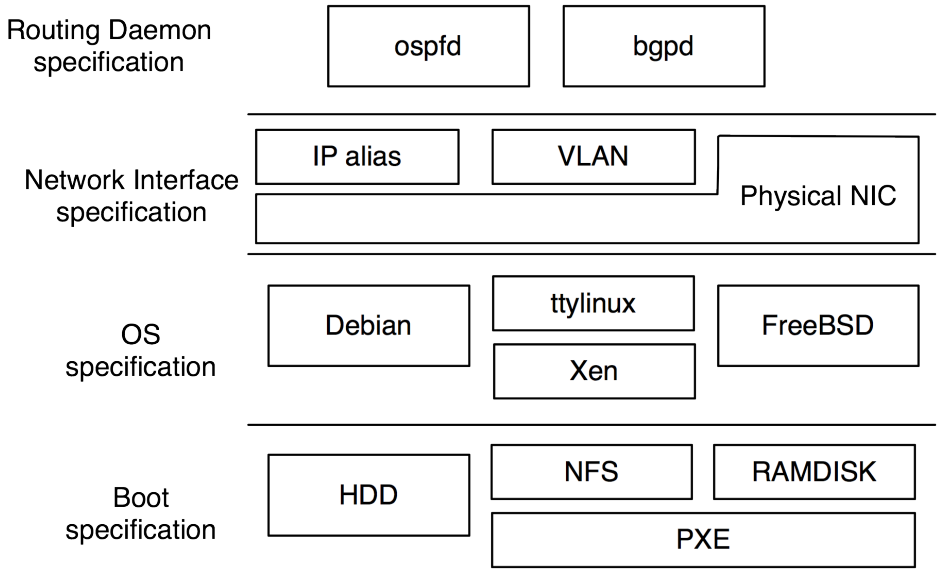

Figure 4.2: Available combination of specifications of AnyBed portable toolset

bringing one master server that packages these toolset into the NETs.

This bringing method aims to support such a use case that users can bring AnyBed-styled master server into the NETs, and can perform experiments with the toolset in the server. Generally, almost all PCs in NETs are PC/AT architecture, and support PXE boot. These PCs are quickly used as AnyBed-styled experimental nodes by the master server without care of installed OSes in node's HDD. Accordingly, I consider that AnyBed toolset realize system portability among various NETs.

Figure 4.2 shows currently available combination of each specification about AnyBed portable toolset. The toolset can be used with node's HDD, PXE + NFS, PXE + RAMDISK as boot method. The user choose suitable OSes such as Debian, ttylinux on Xen, and FreeBSD for his experiment. When the user build network topology, he can choose multiplexing method of networks such as IP alias, VLAN, and bare physical NIC depending on size of the topology and capability of switches on a NET. He can also choose appropriate routing daemons such as ospfd and bgpd for the purpose of his experiment.

There are several use cases, which is shown in table 4.1 using AnyBed with other assistant tools. These use cases show one of the evidences to show modularity and affinity of AnyBed.

Experiment	XENebula	InterTrack	SecureOverray
Scenario Control	kuroyuri	gsh tools	original tool
Node Control	kuroyuri sbpsh ipmitool iLO tool	gsh tools sbpsh ipmitool iLO tool	original tool sbpsh ipmitool iLO tool
Topology (L3)	AnyBed	AnyBed	
Toplogy (L2)	bwsc.pl	bwsc.pl	bwsc.pl
Injection Layer	XENebula sbpsh	AnyBed sbpsh	AnyBed sbpsh
Disk type	NFS RAM HDD	NFS RAM HDD	NFS RAM HDD

Table 4.1: Modulrarity amongs assistant tools

4.3.2 Injection layer tools

In the injection layer, config injector injects actual configuration files to each PC node and switch. All configuration files are classified into two types. One is for PC nodes, and the other is for switches.

The implementation of config injector would be different from each NET environment. For general use with AnyBed, I implement two types of config injector for PC nodes. The first one is using the modified version of "mkdiskimage" from StarBED [1]. The mkdiskimage is the environment for making memory file system image of PXE [32] boot. When a cluster node boots, the node gets two contents: memory file system image from TFTP server and tarball from FTP server. The memory file system image is mounted to root file system, and tarball is extracted there. Then, operating system reads configuration files there. In the tarball, I pack actual configuration files.

The second method injects actual configuration files over NFS. The AnyBed portable toolset previously described uses this method. After a cluster node boots from PXE, the node gets actual configuration files from mounted NFS directory.

For switches, I implement another type of config injector. This method communicates to switches via TELNET [34] and modifies the configuration of each switch. In this version of the program, I assume that only one user uses the switches simultane-

ously. Therefore, the program operate the switches as an administrative user. Aside from TELNET, the standardized mechanisms to modify the configuration of switches is "The Network Configuration Protocol"(NETCONF) [35–38]. NETCONF provides mechanisms to install, manipulate, and delete the configuration of switches. It employs an XML-based data encoding for the configuration data as well as the protocol messages. But, I found a problem on employing NETCONF for the config injector; only enterprise switches can support NETCONF. Low-end L2 switches that are commonly used in lab-level small NETs do not support NETCONF.

4.4 Verification

In this section, I describe the verification result of the AnyBed implementation. I conducted the following verifications and evaluations: topology reusability, system portability, and scalability to the number of nodes.

4.4.1 Topology reusability

In this section, I verify the functions of AnyBed. In the verification, I prepared two PC clusters that their hardware equipment is different from each other. On these two clusters, I built the same network topology by AnyBed to verify topology reusability. Also I compared number of configuration files, the size of configuration files and the average time spend for building the network to verify workload reduction. Then, I verified system portability of AnyBed using another two different PC clusters.

Equipments on each PC cluster are described in Table 4.2 and Table 4.3. The former PC cluster was homogeneous, that is, it was composed of 17 blade servers where each blade server had the same hardware and software spec. On the other hand, the latter PC cluster had heterogeneous nodes, that is, it was constructed with different spec PCs and several vendors' switches.

The homogeneous PC cluster was comprised of 17 blade servers and a layer 2 switch made by Dell. I used one server for DHCP, FTP server, and the other for building experimental network. These cluster nodes were connected with each other by 6 layer2 switches. Because all equipped NICs were PXE-capable, I used PXE boot and mkdiskimage for config injector. The heterogeneous PC cluster was composed of 7 nodes. Node#1 to node#3 and node#4 to node#7 were different in the type of NIC.

Therefore, bandwidth and the name of network interface in operating system were different between two groups. In the heterogeneous PC cluster, I could not use PXE boot for config injector because all equipped NICs were not PXE-capable. Instead of PXE boot, I used NFS for config injector.

On these two PC clusters, I gave the same logical network file that 7 routers were connected with full mesh links, then verified whether the same experimental network topologies were built or not. To investigate network topologies, I logged in to Zebra OSPF daemon via telnet, and got the routing information by executing "show ip ospf route" command [39].

Obtained routing information in both cluster has the same routes and the same routing costs. This result means the same network topology including the same link structure and link costs among routers have been built in both cluster, I consider.

Table 4.2: Verification environment for topology reusability (homogeneous cluster)

Node#1 to Node#17	CPU	Intel Pentium3 1.4GHz
	Memory	1024MB
	NIC	Broadcom BCM5703X(1000Mbps) x 2
Layer2 switch		DELL PowerEdge1655MC Switch

Table 4.3: Verification environment for topology reusability (heterogeneous cluster)

Node#1 to Node#3	CPU	Intel Pentium3 450MHz
	Memory	256MB
	NIC	Intel Pro 10/100B/100+(100Mbps)
		3Com 3c905B-TX(100Mbps)
Node#4 to Node#7	CPU	Intel Pentium3 900MHz
	Memory	256MB
	NIC	3Com 3c905B-TX(100Mbps)
		Netgear GA620(1000Mbps)
Layer2 switch		FoundryNetworks EdgeIron4802F
		ExtremeNetworks Summit48
		ExtremeNetworks Summit5i

Table 4.4: Verification environment for system portability (Cluster#1)

CPU	AMD Mobile Athlon XP 1800+ 1.53 GHz
Memory	2048MB
NIC	Broadcom Gigabit Ethernet Adapter x 2
Operating System	Debian GNU/Linux sid
Layer2 switch	Sun Fire B1600 Switch

Table 4.5: Verification environment for system portability (Cluster#2)

CPU	Intel Pentium3 1GHz
Memory	512MB
Operating System	Debian GNU/Linux sid
Layer2 switch	Cisco Catalyst 6509

4.4.2 System portability

Then, I verified system portability of AnyBed using another two different PC clusters. One cluster had 32 blade servers described in Table 4.4 and another cluster had 32 1U servers described in Table 4.5. At first, after I connected the master server including the AnyBed portable toolset, I built BGP topology on the first cluster using the toolset. Secondly, I disconnected the master server and brought it to the facilities that another cluster was located. Then, after I connected it to another cluster, I had quickly rebuilt the same topology with regenerating actual configuration files. This verification shows that AnyBed toolset has system portability.

According to the results of verifications, I conclude that AnyBed enables a topology to reuse among 4 NETs and that AnyBed can reduce the workload of building experimental network on NET.

The result of evaluation and verification shows that AnyBed has achieved the design goals described in Section 3.1.1. First, Section 4.4.1 has described about the reusability of network topology. Then, I have described the portability of AnyBed in this section.

4.4.3 Researchers using AnyBed for their evaluation

Several independent papers confess that AnyBed helps their evaluation. Iimura et al. [40] used AnyBed with twelve PCs of a Small NET of his lab, and 296 PCs on StarBED to evaluate his Peer-to-Peer Overlay framework for Multi-player Online Games. Masui et al. [41] evaluated his measurement platform named "N-TAP" using AnyBed with 128 PCs on StarBED and PlanetLab nodes. Hazeyama et al. [42] used AnyBed to evaluate their research implementation about network security. They [43] also emulated inter-AS topology that composed of 449 Japanese ASes. These evidences show that AnyBed certainly helps researchers to achieve PDCA cycles on development of large-scale distributed systems.

4.5 Summary

In this chapter, I explored the topology reusability and system portability of AnyBed. Topology reusability was verified in homogeneous NET and heterogeneous NET. System portability was achieved by AnyBed portable toolset. Several researchers employed AnyBed to test their implementations across small NET and a large-scale NET.

Chapter 5

Inter Autonomous System Network Topology Emulation[1]

In this chapter, I present the system to network topology of inter autonomous system (inter-AS). This system is located in the design layer of the whole AnyBed design. To reduce user's workload while generating realistic inter-AS topology, I consider the approach to automatically generate the topology based on observed dataset. AnyBed reduces the size of the topology depending on the number of nodes on target NET, and generates that. I confirm scalability of the implementation.

5.1 Issues of Inter-AS Network Topology Emulation

According to the ISC report [44], the Internet is composed of over 600,000,000 hosts. It is hard and not realistic to emulate all Internet hosts on a NET, even when a large scale NET is available.

Figure 5.1 shows a model of inter-AS network topology. Autonomous System is a unit of routing on the Internet, which advertises and aggregates subnet routes by an Exterior Gateway Protocol, typically by BGP. The current Internet is composed of

[1]This chapter is based on one journal. The journal is "Expediting Experiments across Testbeds with AnyBed: A Testbed-Independent Topology Configuration System and Its Tool Set" in *IEICE Transactions on Information and System*, Vol. E92-D, No. 10, pp. 1877. I edited the paper with focusing on the inter-AS network topology emulation.

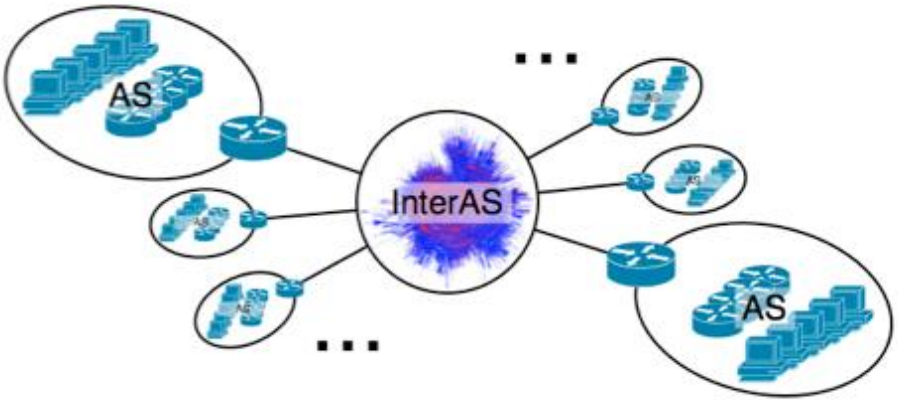

Figure 5.1: Characteristic of Inter AS topology on the Internet

over 33,000 ASes [45]. Many researchers have tried to emulate inter-AS topology to test inter-domain messaging protocols or large scale distributed systems in simulators or NETs. In this chapter, I explore the way of constructing inter-AS topology on a NET with AnyBed.

5.1.1 Approaches On Creating Realistic Experimental Networks

Many researches have explored appropriate realistic topologies to evaluate DDoS countermeasures, BGP behaviors, and so on.

Gong et. al [46] simulated their inter-AS packet traceback with 3 large scale inter-AS topologies to evaluate deployment scenarios of their inter-AS packet traceback protocol. Their first topology was a 8998-node subgraph of an inter-AS topology generated from 3 weeks of CAIDA's AS Relationship dataset collected from 1 to 23 June, 2004 [47]. They also considered a randomly generated an inter-AS topology with 10,000 nodes created by BRITE (2006) [22] using the Barabasi and Albert (BA) model in generating their second topology. They mentioned that both of their inter-AS graphs obey commonly observed power-law degree characteristics of the Internet. Gong's third topology was a 30 $\times$ 30 mesh which were intended to investigate the impact of long paths on the traceback performance.

Zhang et.al. tried to outfit a subgraph of the real Internet onto 72 nodes of DETER testbed [48] to evaluate multi-origin AS (MOAS) attacks. They chose ASes according to tier-level mentioned in [49]. There were three Tier-1 ASes, four Tier-2 ASes and seven Tier-3 ASes. Each Tier-1 AS had three fully connected zebra routers. The three Tier-1 ASes were full meshed. In 4 Tier-2 ASes, two of them multi-home to two Tier-1 ASes, the others only connect to one Tier-1 AS. Tier-3 ASes emulate stub ASes. Carl et.al also tried to construct a subgraph of an inter-AS topology in the DETER testbed to evaluate MOAS attacks [8]. They outfitted 50 ASes subgraph of 22,086 measured ASes by Route Views in April 1, 2006. Chertov et al. developed useful topology generation tools [50] for their experiments on DETER. Their tools can construct Inter-AS and Intra-AS topology based on real topology data obtained by their tools and Route Views project. Their tools can pick up a set of ASes from the dataset, or can perform breadth-first traversal of the topology graph from a specified AS number.

Jin et al. developed "*inet*", an AS-level Internet topology generator [51]. Based on BGP table datasets by RouteView Project [52], the inet generates random networks with characteristics similar to those of the Internet from November 1997 to June 2000, and beyond. The size of a generated topology by inet cannot be no less than 3037 nodes due to the size of base dataset that is the number of ASs on the Internet in November 1997.

Hazeyama et al. proposed four filtering rules for CAIDA's AS Relationship dataset (ASDR) in [42] to outfit subgraph of an inter-AS topology to available NET resources. Four filtering rules in [42] are *Top-Ranking Filtering (TRF)*, *Root-AS Neighbors Filtering (RANF)*, *Region-Based Filtering (RBF)* , and *List-Based Filtering (LBF)*.

Hazeyama also pointed "*Isolated Island Problem*" and "*Overflow Problem*" on outfitting ASDR inter-AS topology to available physical resources. Isolated Island Problem is that a subgraph of an inter-AS Topology, which is composed of selected Top N ASes, included an AS that has no neighbor in the subgraph, or a subgraph is separated into several pieces. Because of the complexity on the peering by policy, several Top rank ASes don't peer due to the competitor. Two competitors are connected by one middle rank AS which will not be included top N ASes. On the other way, Overflow Problem is that the select number of ASes by a filtering rule is easily larger than the number of available physical resources. Overflow Problem is easily occurs RANF. To avoid Isolated Island Problem and Overflow Problem, Hazeyama also proposed a fil-

tering manner which is combined to several filtering rules to fit an inter-AS topology to available physical resources.

5.1.2 AS Relationship Dataset

AS Relationship Dataset is a dataset which describes of adjacency among ASes. CAIDA Project has produced IPv4 AS Relationship Dataset(ASRD) [47]. Supported by Route Views Project [52], CAIDA Project measured BGP4 full route information in several backbones and analyzed an inter-AS topology according to an inferring method mentioned in [53]. Also, State Key Lab. [54] has measured IPv6 AS topology data.

An ASRD shows the state of each link between two active ASes. Variation of link state is determined by an algorithm described in [53]; *provider link*, *customer link*, *p2p link* and *siblings link*. Around 7th, Jan., 2008, there were 26,961 IPv4 ASes announced by 2 byte AS Number (ASN), and 583 IPv6 ASes existed. These link information were announced by BGP or BGP+.

5.1.3 Model of Inter-AS Network Topology on NET

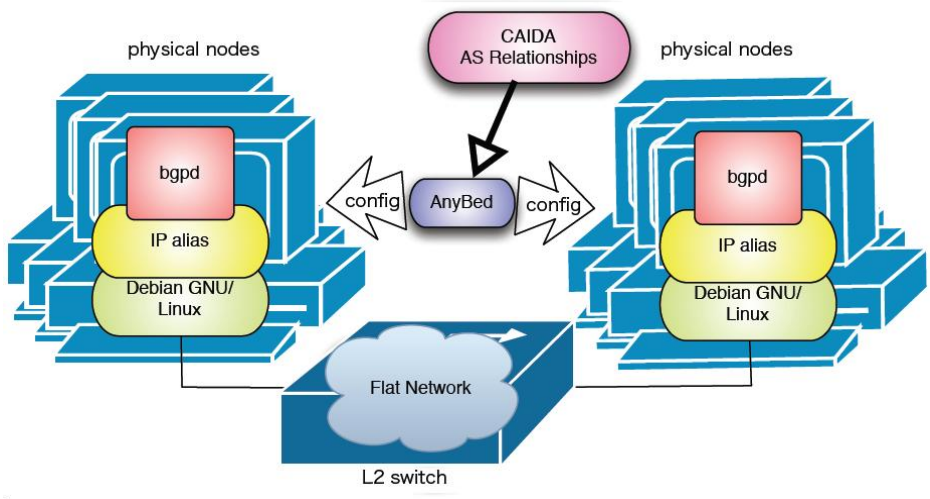

Figure 5.2: Model of Inter-AS emulation with AnyBed

Figure 5.2 shows the emulation model of inter-AS network with AnyBed. In inter-AS topology emulation, the number of VLAN ID is easily consumed. Different from

OSPF, BGP is independent from the layer 2 link, therefore, BGP routing can be achieved on virtual interfaces created by IP alias shown in Fig. 5.2. On inter-AS topology emulation, AnyBed employs IP alias ID for virtual interfaces instead of VLAN ID.

5.2 Implementation

In this section, I describe the second implementation of assignment layer, which is the main component of AnyBed. I implemented that in Ruby 1.8.6.

Dispatcher on the assignment layer reads both a physical network file and a logical network file, assigns the elements of the physical network topology to the elements of the logical network topology, and makes an experimental network topology with physical and logical information. Then, *config generator* converts the experimental network topology to actual configuration files along with the syntax of each software or each switch. In injection layer, *config injector* injects those actual configuration files to each PC node and switch.

5.2.1 Design layer tools

I redesign the logical topology and re-implement some tools used for the design layer. First one is a description converter to a physical network file. This converter converts the resource description file used by StarBED's assistant tools named "SpringOS" [55] to the format of AnyBed.

Second one is filter scripts named CAIDA topology filter [42]. These scripts enables me to pick up proper size of AS relationship data from whole dataset obtained from CAIDA project [47]. CAIDA Project has measured BGP4 full route information in several backbones and analyzed an inter-AS topology according to an inferring method [53]. This AS relationship dataset shows the state of each link between two active ASes. These picked data are finally converted into a logical topology file of AnyBed. Using this file, users can easily build an inter-AS BGP topology like as MOAS experiments topologies [8].

Third one is a script converting a logical network file to Graphviz dot file [56]. Graphviz is open-source graph visualization software. Users can visualize and confirm their designed topology by using Graphviz with this script.

5.2.2 Configuration files for EBGP Network Topologies

In this section, I explain the design of the logical network file for EBGP :

A logical network file describes a logical network topology that the user desires to build.

The example of a logical network file for EBGP is shown in Figure 5.3. Figure 5.3 shows a part of a small BGP topology.

Here I explain the syntax of the first example. In the first level, a `<nodes>` presents for node sets. A `<nodes>` has some `<node>` that presents an experimental node. A `<node>` has some `<interface>` that present network interfaces on each experimental node. A `<interface>` has some `<network>` that present network on each interface. A `<function>` presents that the node as routing functions line OSPF or BGP.

On `<bgp>` of `<function>`, `<advnetworks>` contains `<advnetwork>` which represent a subnet network advertized by BGP to neighbor ASes. `<neighbors>` has sub elements `<neighbor>` which shows a BGP neighbor AS. The attribute "asname" of `<neighbor>` shows neighbor AS's name, "type" represents peering protocol (EBGP or IBGP). The attribute "relationship" has 4 type values, "peer", "customer", "provider", and "siblings". According to the value of "relationship", AnyBed generates a simple route map figured in Fig. 5.4 which is a set of filtering rules for advertized routes and AS paths. The attribute "localint" and "remoteint" represent peering interfaces. "localint" should be listed on `<interface>` of the current `<node>`, on the other hand, "remoteint" should appear on `<interface>` of another `<node>` part.

5.2.3 Extension of the Assignment layer

Another choice for the user is building a network "IP alias mode" of config generator. In this mode, network topology can be constructed by assigning alias IP address instead of using VLAN mentioned in section 3.2. In the assignment layer, dispatcher reads a physical network file and a logical network file. Then, dispatcher assigns physical elements to logical ones properly. The physical elements are physical nodes, network interfaces in the nodes, IP addresses, IP alias ID and bandwidth. In the current implementation, the criterion for properness is only fairness of bandwidth of each physical link. In a logical network file, there is a tree structure: "node" - "interface" - "network". Similarly, in physical network file, there is a tree structure: "node" - "interface" - "link".

```
<nodes>
  <node name="AS65001">
    <interfaces>
      <interface name="Int-AS65001">
        <network name="Net-AS65001"/>
      </interface>
      <interface name="Int-AS65001-AS65002">
        <network name="Net-AS65001-AS65002"/>
      </interface>
    </interfaces>
    <function>
      <ospf>
      </ospf>
      <bgp name="AS65001"  asname="AS65001" asnum="65001">
        <advnetworks>
      <advnetwork name="Net-AS65001"/>
        </advnetworks>
        <neighbors>
          <neighbor asname="AS65002" type="EBGP"
           relationship="peer"
           localint="Int-AS65001-AS65002"
           remoteint="Int-AS65001-AS65002"/>
        </neighbors>
      </bgp>
    </function>
  </node>
</nodes>
```

Figure 5.3: Example logical network topology using BGP

```
ip as-path access-list MyAS permit ^$

ip community-list standard Customer-RT permit 65001:200

route-map import-from-customer permit 10
 set local-preference 200
 set community 65001:200

route-map import-from-peer permit 10
 set local-preference 100

route-map export permit 10
 match as-path MyAS

route-map export permit 20
 match community Customer-RT

route-map export deny 30

route-map export-my-route-only permit 10
 match as-path MyAS

route-map export-my-route-only deny 20
```

Figure 5.4: Example of quagga bgpd route map in peer case

Algorithm 2 An algorithm of assigning nodes and interfaces

```
 1: procedure Assignment Main Routine
 2: sort(LogicalNodes) order by Number_of_Interfaces
 3: sort(PhysicalNodes) order by Number_of_Interfaces
 4: sort(LogicalNodes.Interfaces) order by Bandwidth
 5: sort(PhysicalNodes.Interfaces) order by Bandwidth
 6: for all LogicalNodes do
 7:     l ← LogicalNode.Interfaces
 8:     p ← PhysicalNode.Interfaces
 9:     for all l do
10:         p.ResidualBandwidth ← p.TotalBandwidth
11:         MaximumInterface ← maximum(p) order by Bandwidth
12:         MaximumInterface.AliasID
            = assignNewAliasID(MaximumInterface)
13:         AverageBandwidth ← p.TotalBandwidth / l.TotalBandwidth
14:         p.ResidualBandwidth ← p.ResidualBandwidth - AverageBandwidth
15:         remove MaximumInterface from l
16:     end for
17: end for
```

Dispatcher reads these structures, and then make assignment based on the algorithm shown in Algorithm 2. The reason that I adopt the simple iterative algorithm is that low computational effort is more important than bandwidth optimality. If I seek bandwidth-optimum, I must solve knapsack problem. On the large scale NET, solving knapsack problem costs high computational effort, and makes the user wait for a long time. To improve efficiency of this assignment, I must study more about a heterogeneous hardware case of "network testbed mapping problem" [57]. It is not the target of this paper however.

After dispatcher finished assigning of nodes and interfaces, dispatcher assigns IP addresses to experimental interface according IP alias ID, and an experimental network topology is generated as the result of resource assignment by dispatcher. Then, config generator translates from the topology to actual configuration files.

The variations of actual configuration files are as follows: rc.conf, hosts, zebra.conf, ospfd.conf and switch configuration files of each vendor's syntax.

5.2.4 Consistency checker

To check whether the desired topology is correctly built or not, I made consistency checker as a part of AnyBed. The checker reads output files from AnyBed, then check reachability from all nodes to all nodes, and also check routes in all nodes. Figure C.1 shows a result of the reachablility check(upper) and a result of the routecheck(lower). The vertical axis in this matrix means nodes that check rearchabilities and routes. The horizontal axis means checked nodes and checked routes. Green means reached, while grey means unreached.

5.2.5 Building experimental Inter-AS network using AnyBed

The steps for the user to build experimental network with AnyBed is shown in Figure 3.3. The steps are described below:

1. A NET prepares a physical network file along with its facilities and wiring.

2. A user designs a logical network topology and creates a logical network file. CAIDA topology filter facilitates generating this file. Of course, the users can edit the file manually.

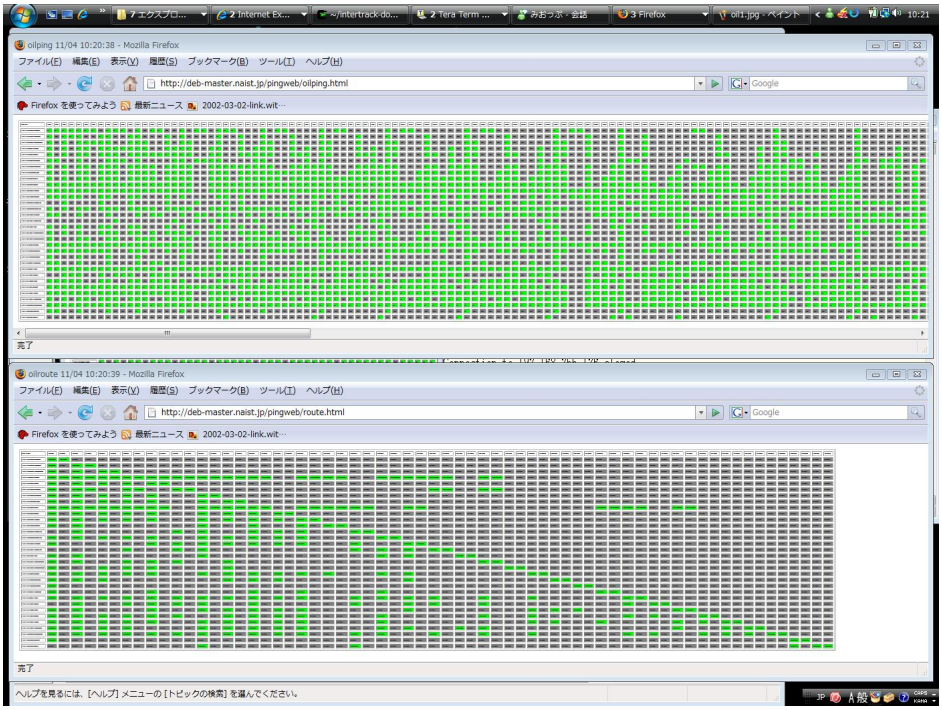

Figure 5.5: Results of rearchability check and route check

3. The assignment layer of AnyBed checks the consistency of the logical network file and the physical network file.

4. The assignment layer assigns resources to the logical network topology and generates actual configuration files, if there is no inconsistency between the logical network file and the physical network file.

5. The injection layer of AnyBed injects those actual configuration files to each node and switch.

Compared to the manual steps described in Section 2.5.1, AnyBed and its toolset automate most processes: some parts of design, assignment, configuration, and injection. The detail evaluation and verification are described in the following sections.

5.3 Evaluation

In this section, I describe the evaluation result of the AnyBed implementation. I conducted the following evaluations: scalability to the number of nodes and workload reduction.

5.3.1 Scalability to the number of nodes

In this section, I see if AnyBed can deal with large scale network and many nodes. Because I do not have a real large scale NET, I use StarBED for my scalability evaluation; which is the well-known large-scale centralized NET that composed of 680 nodes. I describe evaluation environment. I used 150 experimental nodes on StarBED and 1 master node. As for a typical user on StarBED, he uses about 50 up to 150 nodes for one experiment. In DETER, recent experiments [8, 58, 59] need roughly 10 to 100 nodes. I think that evaluating AnyBed using up to 150 nodes is enough for typical uses. The specifications of nodes are described in Table 5.1.

In this evaluation, I described the logical network file where nodes had BGP links each other. This BGP topology was based on the AS topology of the Internet using the dataset published by CAIDA [47]. I extracted top ASes and its related networks from this dataset in order of the number of BGP peers that ASes have. The number of extracted ASes depended on the number of usable nodes in NET. Using this topology,

I made an experimental network and measured the time consumption with changing the number of nodes from 10 to 150 on the environment. In this evaluation, because 1 AS in the dataset corresponded with 1 node in NET, I could only emulate a part of the whole topology. I measured times taken by below procedures. (1) I ran dispatcher on the master node to generate actual configuration files, and then deployed them to experimental nodes. (2) After deployed, I set interface of the experimental nodes up. (3) Finally, I set BGP connections of each node up.

Table 5.1: Evaluation environment for scalability

Master Node	CPU	Intel Core 2 Duo 2.13GHz
	Memory	1GB
	Operating System	Debian GNU/Linux sid
	Programming Language	Ruby 1.8.6
Exp. Nodes	CPU	Intel Pentium3 1GHz
	Memory	512MB
	Operating System	Debian GNU/Linux sid
Layer2 switch		Cisco Catalyst 6509

Figure 5.6 shows that the total time consumption increased linearly up to 150 nodes. However, the time generating configuration files and deploying them does not seem to linearly but exponentially. The reason that the curve described in the exponential function was that the BGP topology was nearly full meshed and the number of subnets and interfaces increased in the order of $O(n^2)$. The number affected the time of assigning resources and the time of generating configuration files, I considered. On 150 nodes, the total time consumption of dispatcher was 113 seconds.

Along with this evaluation, I conclude that (1) AnyBed can ease the overhead of describing configuration files in the order of magnitude, and (2) it can generate configuration files and can deploy them on a large scale NET in enough short time.

5.3.2 Decline of routing performance

In this section, I discussed the decline of routing performance when a user employs AnyBed. I evaluated the differences of routing performance on two same topologies: one was built by AnyBed, and the other was built by hand. Figure 5.7 shows these topologies. On these topologies, I assigned the same PC for each node. Equipments on

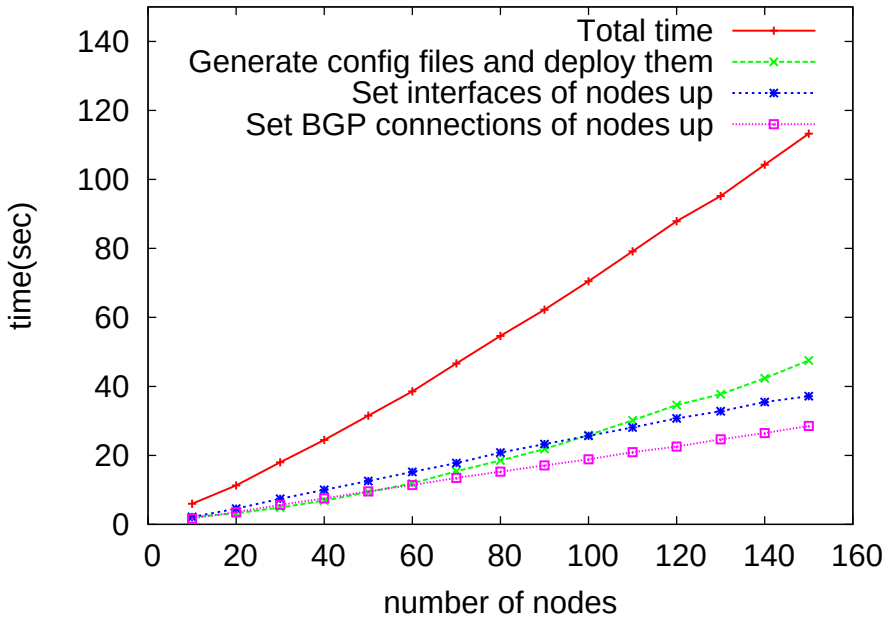

Figure 5.6: Time consumption of building experimental network under different number of nodes

each PC are described in Table 5.2. On each topology, I measured routing performance 10 times from node1 to node6 by iperf.

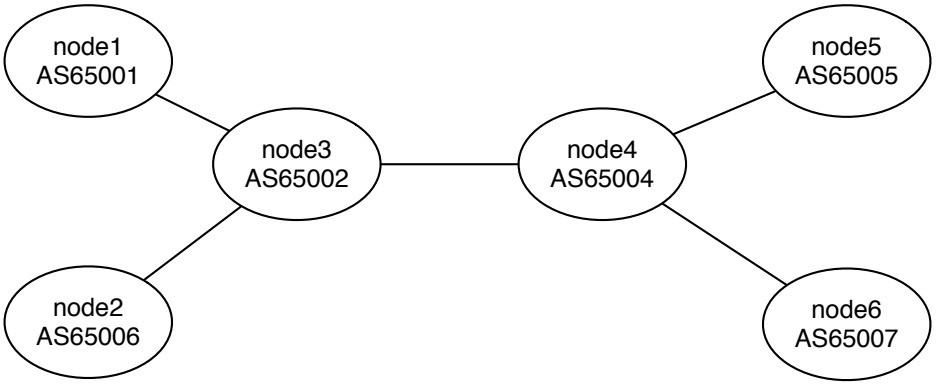

Figure 5.7: Evaluation topology for routing performance

The result of the measurement showed that both topologies had the same routing performances. In both cases, measured routing performance was 914Mbps. The result means that AnyBed does not cause decline of routing performance.

5.4 Summary

As many researches have explored appropriate realistic topologies to evaluate DDoS countermeasures, BGP behaviors, and so on, inter-AS topology emulation was employed in their evaluations In many cases.

To achieve inter-AS network topology emulation by AnyBed, I modeled inter-AS topology emulation on NET, re-design the XML schema of the logical topology file to support BGP function, re-implemented assignment layer and injection layer of AnyBed to cooperate with CAIDA ASRD. I verified and evaluated new features of the assignment layer implementation.

The results of the verification and evaluation showed that AnyBed well automated the steps to construct an inter-AS network topology on various NETs. AnyBed can expedite the time spent building 50 AS topology, from 4 days in manual to 30 seconds in AnyBed.

Table 5.2: Evaluation environment for routing performance

Master Node	CPU	Intel Xeon 2.13GHz
	Memory	4GB
	Operating System	Debian GNU/Linux etch
	Programming Language	Ruby 1.8.5
Exp. Node1,2	CPU	Intel Xeon 2.8 GHz
	Memory	2.5GB
	NIC	Broadcom GbE NIC x 2
Exp. Node3	CPU	AMD Opteron 1.8 GHz
	Memory	4GB
	NIC	Broadcom GbE NIC x 2
Exp. Node4	CPU	Intel Xeon 2.8 GHz
	Memory	2GB
	NIC	Broadcom GbE NIC x 2
Exp. Node5	CPU	Intel Xeon 2.8 GHz
	Memory	2GB
	NIC	Broadcom GbE NIC x 2
Exp. Node6	CPU	Intel Xeon 2.8 GHz
	Memory	1GB
	NIC	Broadcom GbE NIC x 2
Layer2 switch		Buffalo LSW-GT-16NSR

Chapter 6

OSPF Network Topology Emulation[1]

In this chapter, I present an OSPF network topology emulation system to emulate the topology of intra autonomous system(intra-AS). This system is located in the design layer of the whole AnyBed design. Firstly, I analyze issues of intra-AS topology emulation, and propose the design of the system. Then, I describe the implementation and the evaluation of the system.

6.1 Issues of Intra-AS Topology Emulation

On the contrary of dataset of inter-AS topology, the dataset of intra-AS topology usually does not become public dataset due to security issues on daily network operation. However, detail information such as cost settings can be available from an intra-AS network for private use if a NET user operates the intra-AS network.

In this chapter, I explore the way of more realistic network topology emulation about actual operated network in focus on OSPF network.

[1]This chapter is based on one domestic conference paper. The domestic conference paper is "A Topology Imitation System from A Real OSPF topology to A Network Emulation Testbed" in *Proceedings of Internet Conference 2009 (IC2009)*, Octobor, 2009. I edited these papers with focusing on the OSPF network topology emulation with an actual operated OSPF network.

6.2 Design of ONTES: an OSPF Network Topology Emulation System

When I design ONTES, I have several considerations of network topology collection methods and emulation methods. If I can obtain all configurations from all routers in the target network topology as dataset, I can emulate a network topology completely. However, the configurations for routers are generally secret because they are sensitive information in network operation. Even configurations of routers are can be collected, the cost of implementing the mechanism to collect all configurations from all routers is high to support multi-vender implementations and differences of each vendor syntax. Also, the configurations of routers include unnecessary information to construct an emulated network topology.

Many researchers infer topology and configuration by various approaches, and create their own datasets that contain suitable information for their purposes. In the datasets, the unnecessary information except for their purposes, such as interface cost settings and IP addresses of each interface are mostly degraded or divided into another dataset. The information contained in public dataset is insufficient for my purpose to construct accurate topology. I need methods for efficient collection and accurate topology emulation.

I define design goals described below.

Goal 1. To collect information for private use from actual operated network

Goal 2. To emulate OSPF topology

Goal 3. To collect the information as efficient as possible

Goal 4. To emulate the topology as accurate as possible from the collected information

Goal 5. To shorten the time elapsing from collecting to emulating as possible

About Goal 1 and Goal 2, I consider the middle-size network as that consists of several to dozens of routers. This number of routers depends on my first emulated target of ONTES; my research organization named WIDE Project [60] has the OSPF network composed of approximately 49 routers around Japan.

AnyBed previously adopted private IP addresses on constructing BGP network because I focused on not actual assigned IP addresses but macroscopic AS relationships.

However, in case of emulating OSPF network, I aim to emulate actual operated network information that includes assigned IP addresses because this information would be important to accomplish the purpose of this research.

Goal 5 comes from the policy of AnyBed that the time users can perform their experiment should be as long as possible. That is because, on a general large-scale NET in Japan, users can use the facility only for a limited period on reservation basis. I intend the preparation time until an user can perform an experiment to be as short as possible. This time contains collecting targeted network information, assigning resources, generating configuration files, distributing them, and checking. Previous AnyBed implementation also aimed to meet this requirement.

6.3 Implementation of ONTES

Related with Goal 1 and Goal 3, I come up with three approaches to collect.

Appr.1. Inferring from the results of traceroute from various points to other points

Appr.2. Inferring from the Routing Information Bases(RIBs) and Forwarding Information Bases(FIBs) obtained from all routers that compose the OSPF network

Appr.3. Inferring from the OSPF link-state database(OSPF LSDB) obtained from one router in the OSPF network.

I adopt Appr.3, and describe the reasons below. Appr.1 is the similar approach to Rocketfuel. This approach can only collect the best route, cannot collect the whole network topology information including OSPF cost. I do not adopt this approach because of the disadvantage of the accuracy of the collected topology. About Appr.2, in case that a network is composed of multi-vendor routers, each router has its own command syntax to get RIB and FIB, and low-cost routers do not support OSPF MIB in SNMP. The implementation cost that covers these multi-vendor differences is high. I dismiss this approach.

Finally, when I adopt Appr.3 because of efficiency. There are still two approaches for obtaining LSDB; one is by TELNET, and the other is by SNMP. I adopt the former because this approach is expected to obtain other information than LSDB versatility in the future. Compared to Appr.2, the implementation cost is lower because the number of vendors that I should support is only one.

Algorithm 3 OSPF Topology Assumption

```
1: procedure OSPF Topology Assumption
2: load Router_LSAs into RLSA_LIST
3: load Network_LSAs into NLSA_LIST
4: for all rlsa from RLSA_LIST do
5:     select link_state_id from rlsa
6:     select TRANSIT_INTERFACE_LIST from rlsa by Link_Type
7:     for all transit_interface from TRANSIT_INTERFACE_LIST do
8:         select address_of_the_interface, cost, designated_router_address from transit_interface
9:         select nlsa from NLSA_LIST by pairs(designated_router_address, address_of_the_interface)
10:        select netmask from nlsa
11:        calculate network_address from pairs(address_of_the_interface, netmask)
12:        output link_state_id, address_of_the_interface, network_address, netmask, cost, designated_router_address
13:    end for
14:    select STUB_INTERFACE_LIST from rlsa by Link_Type
15:    for all stub_interface from STUB_INTERFACE_LIST do
16:        select network_address, netmask, cost from stub_interface
17:        calculate first_address from pairs(network_address, netmask)
18:        output link_state_id, first_address, network_address, netmask, cost
19:    end for
20: end for
```

In Appr.3, I infer OSPF network topology from LSA Type 1(Router LSA, RLSA) and LSA Type 2(Network LSA, NLSA). The detailed algorithm is shown in Algorithm 3. I also briefly describe the algorithm below.

1. To infer network which connect to one router, I obtain the interface addresses of the router(Addrs-A), OSPF costs and the IP addresses of the designated routers(DR-Addrs) from RLSAs which is related with the interfaces connected to transit network.

2. I correlate the router and network by obtaining the network address from an interface address and a netmask from NLSA for each DR-Addrs related with the network that includes each Addrs-A.

3. Similarly, I obtain stub-network addresses and netmasks from NLSA that interface address is included in stub-network connected to the router from RLSA. Due to an interface address of the router is not included in LSAs, I assume first IP address of the stub-network to be the interface address.

After inferring the topology from this algorithm, ONTES write the topology in the extended format of the logical network file used in AnyBed. I show an example of the

extended format in Figure 6.1. This format contains several information about OSPF network such as addresses of interfaces that one OSPF router has, OSPF cost of the interfaces, and a router-id of the router.

```
<nodes>
  <node name="192.168.255.108">
    <interfaces>
      <interface
       name="Node192.168.255.108-Int10.0.7.0.24"
       interfaceaddress="10.0.7.2"
       netmask="255.255.255.0" ospfcost="6">
        <network name="Net10.0.7.0.24"
         networkaddress="10.0.7.0"
         netmask="255.255.255.0"/>
      </interface>
      <interface
       name="Node192.168.255.108-Int10.0.3.0.24"
       interfaceaddress="10.0.3.1"
       netmask="255.255.255.0" ospfcost="5">
        <network name="Net10.0.3.0.24"
         networkaddress="10.0.3.0"
         netmask="255.255.255.0"/>
      </interface>
      <interface
       name="Node192.168.255.108-Int10.0.6.0.24"
       interfaceaddress="10.0.6.2"
       netmask="255.255.255.0" ospfcost="7">
        <network name="Net10.0.6.0.24"
         networkaddress="10.0.6.0"
         netmask="255.255.255.0"/>
      </interface>
    </interfaces>
    <function>
      <ospf routerid="192.168.255.108">
      </ospf>
    </function>
  </node>
  ...
```

Figure 6.1: Sample logical network file

Based on this logical network file, the assignment layer of AnyBed assigns resources and generates configuration files for assigned nodes. On this assignment, one router in the original topology is associated with one node., and multiple interfaces in the router are associated with one physical interfaces on the node using IP alias mechanism. The

types of these configuration files are divided into two parts: one is related with network settings of each OS, and the other is about applications such as ospfd. I show the former example in Figure 6.2, and the latter one in Figure 6.3.

```
hostname 192.168.255.108-192.168.255.108
ifconfig eth1:5 inet 10.0.7.2 netmask 255.255.255.0
ifconfig eth1:2 inet 10.0.3.1 netmask 255.255.255.0
ifconfig eth1:7 inet 10.0.6.2 netmask 255.255.255.0
```

Figure 6.2: Sample configuration file for OS

```
hostname 192.168.255.108
password sample
enable password sample
interface eth1
 ip ospf cost 5 10.0.3.1
 ip ospf cost 6 10.0.7.2
 ip ospf cost 7 10.0.6.2
 ip ospf priority 2 10.0.3.1
 ip ospf priority 5 10.0.7.2
 ip ospf priority 7 10.0.6.2
router ospf
router-id 192.168.255.108
network 10.0.7.0/24 area 0.0.0.0
network 10.0.3.0/24 area 0.0.0.0
network 10.0.6.0/24 area 0.0.0.0
redistribute bgp
```

Figure 6.3: Sample configuration file for ospfd

To realize ONTES, I newly implement a script named ospfwalk to collect data from an actual operated router, and extend the assignment layer of previous AnyBed implementation. The data flow of ospfwalk is shown in Figure 6.4. Firstly, ospfwalk login to the specified router, and issue a command like "show ip ospf database router/network" to get OSPF LSDB. In case the host on which ospfwalk runs cannot directly access to the router, ospfwalk can read the text file that contains the output of the command. Then, ospfwalk infer a topology by the algorithms described above, and write a logi-

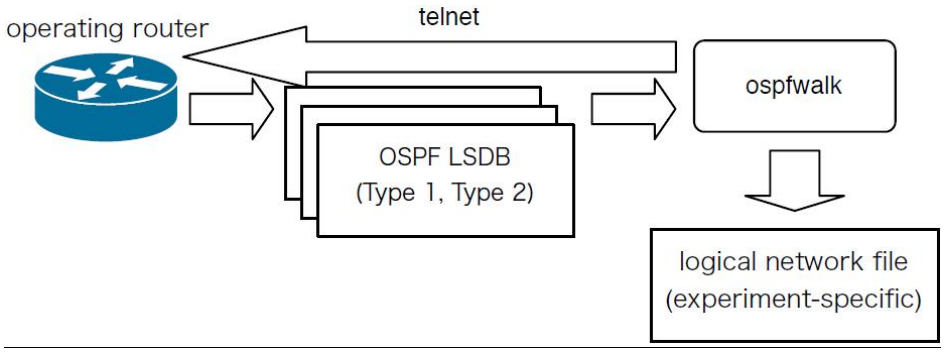

Figure 6.4: Data flow of ospfwalk

cal network file. I also implement a visualization mechanism using GraphViz [56] in ospfwalk because I cannot understand a structure of written topology in text format.

AnyBed cannot accurately emulate Point-to-point(PtP) link because AnyBed uses Ethernet to construct network. Alternatively, AnyBed assigns private IP address to the both side of PtP link, and construct the network. Because I consider the accuracy of a topology structure is more important than that of IP addresses.

Also, I extend the assignment layer of previous AnyBed implementation to generate configuration files that contain actual assigned IP addresses and OSPF costs.

6.4 Verification of ONTES

I verify ONTES in two steps. In the first step, I try to emulate a small OSPF network constructed manually on my NET. Then, I try to emulate actual operated OSPF backbone network of WIDE project. I describe the details of this emulation below.

6.4.1 Emulation of a small OSPF network

In the first step of the verification, I manually construct a small OSPF network(Net.A) composed of 6 routers on my NET, and try to emulate Net.A as Net.A' on the same NET.

I show the structure of Net.A in Figure 6.5. To design Net.A, I complicate a dumbbell topology depending on the number of nodes existing in my NET. All network have private IP addresses with own /24 subnet. Link costs are dispersed for the

verification of emulating that.

In this verification, on Net.A and Net.A', I observe FIBs and RIBs of each corresponding router, and observe the results of running traceroute from R1 to all other routers shown in Figure 6.5. The result shows that FIBs, RIBs and the results of traceroute are all same values. Additionally, I verify the interfaces of corresponding router have the same IP addresses and the same OSPF cost. These results show that ONTES can emulate the same network topology including the same link structure, the link costs and the same IP addresses on a NET.

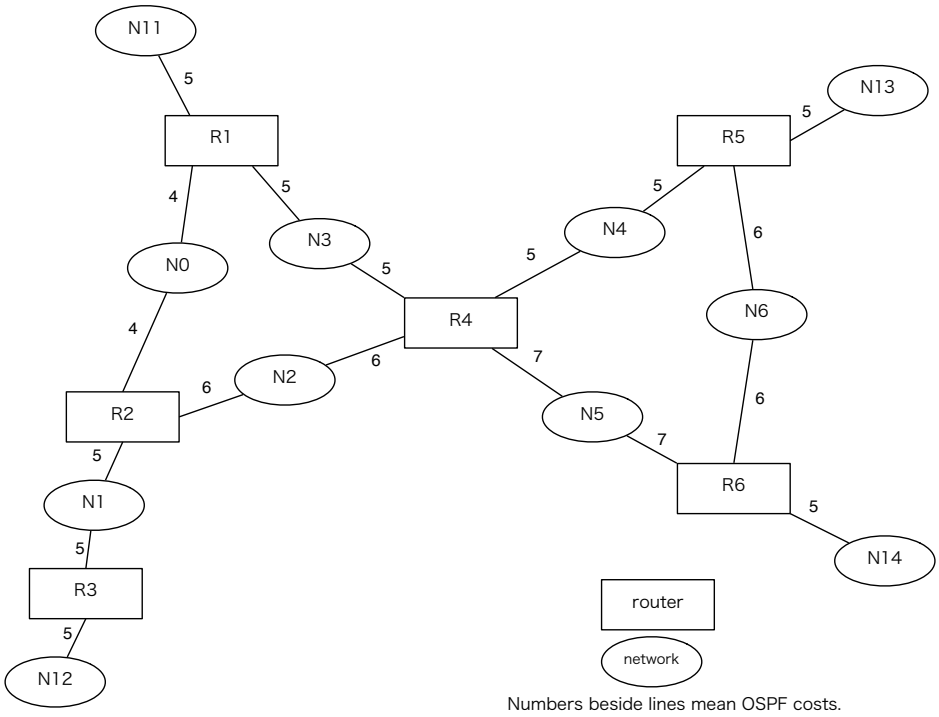

Figure 6.5: Small OSPF network for verification

Consumed time from collecting Net.A to finish constructing Net.B is totally 44 seconds, in the condition that I have prepared installing and booting OS on each node before. This time means enough reasonable for users for NETs, I suppose.

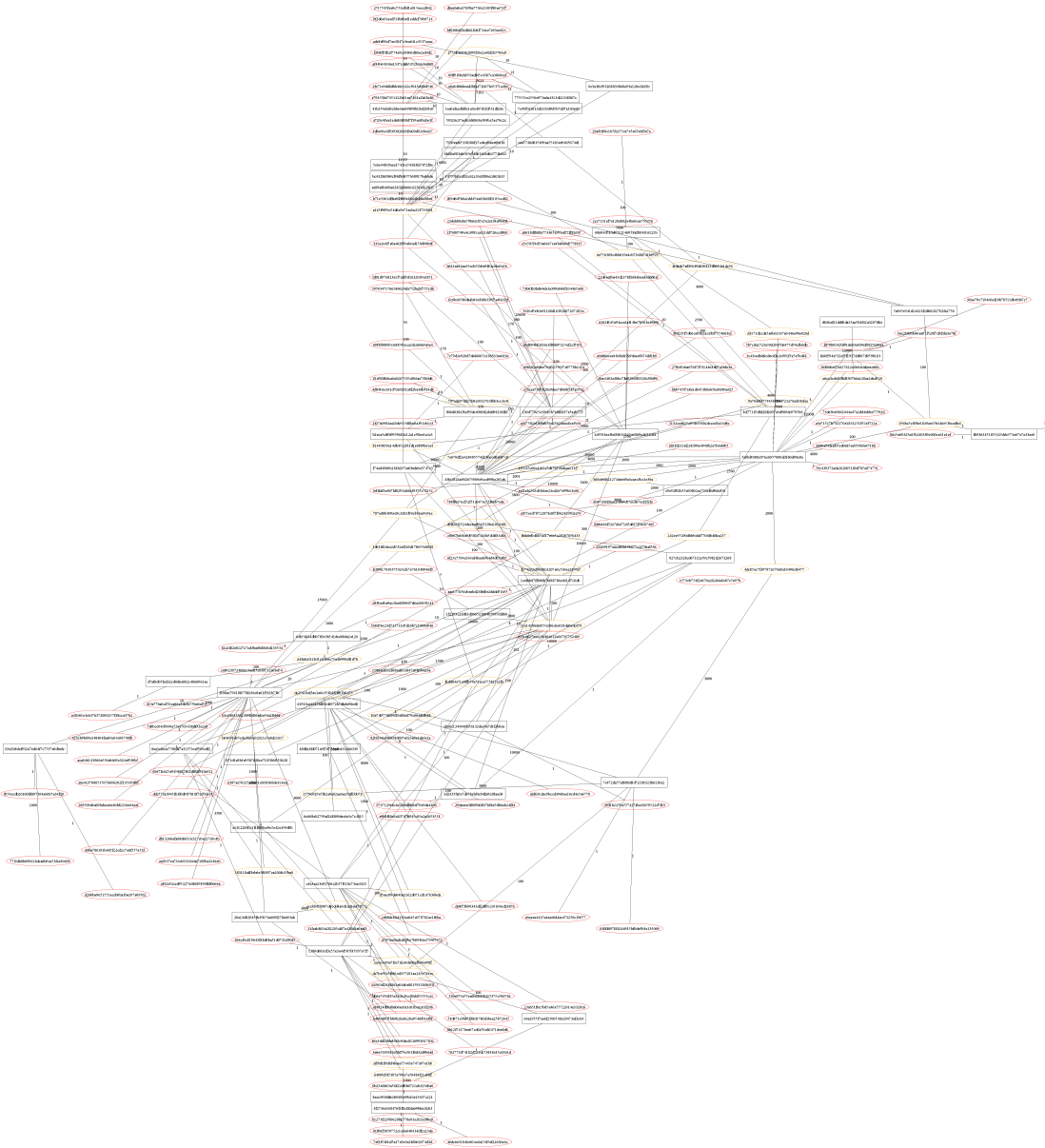

Figure 6.6: Emulated WIDE backbone OSPF network topology

6.4.2 Emulation of a large operated OSPF network

In the second step, I try to emulate the actual operated OSPF network of WIDE project on StarBED. Though the targeted network is the large distributed OSPF network consisted of multiple OSPF areas, I only emulate the backbone area(Net.B) of whole network due to my system limitation of current implementation. Even in the backbone area of the network, 49 multi-vendor routers and 237 network are distributed around Japan. I show the structure of Net.B in Figure 6.6. In this figure, shown IP addresses of routers are hashed because they are nondisclosure.

I collect the information of Net.B from the service host which runs GNU Zebra 0.95 [39] located in Keio University, Japan. Because I cannot connect from the AnyBed-running server in StarBED to this service host, I login to the host via SSH, and do telnet to the console of the ospfd. After logging in, run commands such as "show ip ospf database router/network", and save the result to text files. Then, I input these files to ospfwalk in the AnyBed server. Finally, I construct the emulated network(Net.B') using 49 real nodes in StarBED.

After the construction, I manually confirm differences between the RIBs and FIBs of original 5 router picked from Net.B and corresponding emulated 5 routers in Net.B'. The confirmation result shows that the routes of the original routers includes the routes of corresponding ones, and the number of routes is different each other. The costs of included routes from the routers are the same value between the original routers and corresponding ones. I suppose this result means that the original routers run other protocols than OSPF, and routes from other protocols are imported to RIBs and FIBs.

This verification result shows that ONTES can emulate the topology of links between the original routers, its OSPF costs, partially its IP addresses and the RIBs/FIBs originated from them in the targeted OSPF backbone-area.

6.5 Discussion and future work

In this section, I discuss current problems of ONTES based on the results of verification.

6.5.1 About current collection method of OSPF network information

By current collection method, I cannot obtain detailed information other than the area where obtained router belongs. I can only obtain summary LSAs that contain little information of other areas. I plan to solve this problem by combining information obtained from multiple routers that belong to different areas.

I cannot collect the routes distributed from other protocols than OSPF in current method. I consider that ONTES would collect RIBs and FIBs from all routers, and combine them like Appr.2 in Section 6.3 to solve this problem.

Current collection method cannot determine IP addresses of router interfaces that connect to an OSPF stub-network. Collecting interface information from such router via logging in or SNMP can determine these IP addresses.

6.5.2 About limitations of an emulated network by ONTES

Current constructing method cannot emulate Point-to-point link contained in the original network topology due to Ethernet limitation. I plan to use the function of tunneling interface equipped in OS to solve this problem.

In case of emulating geographically distributed OSPF network, emulating the latency of each link is also important. Technically, I can use delay simulators like netem [24] on each router to emulate the latency. However, the collected information by ospfwalk does not contain the latency. To collect the latency, I would adopt traceroute-based method like Appr.2 in Section 6.3.

6.5.3 Use cases of ONTES

ONTES can emulate the original network topology, FIBs/RIBs on the original routers, and OSPF cost on original links, but cannot emulate the performance of the routers, bandwidth of the links, and delay of the links. This result means that ONTES can be used for experiments that concerned about the topology. An advantage of ONTES is that users can run their software implementation on each emulated router. I consider that the use cases of ONTES are evaluations of software that depend on topology, design of network, behavior observations of network fault, and training for operators.

6.6 Summary

In this chapter, I describe my OSPF network topology emulation. ONTES, which is based on AnyBed, can emulate an actual operated OSPF network topology inside an AS. I explain the design and the implementation of ONTES, and describe the verification results. The results show ONTES can emulate both small OSPF network consisted of 6 PC-routers and actual operated OSPF network consisted of 49 multi-vendor routers.

Chapter 7

Discussion

In this chapter, I try to evaluate my studies in chapter 3.1 to chapter 6 and discuss various pertinent aspects of these studies. First, I compare the latest AnyBed implementation with other NET assistant tools.

7.1 Comparison between AnyBed and Other NET Assistant Tools

Since the beginning of the development of AnyBed in 2003, I have improved AnyBed from a number of varied perspectives. Other NET assistant tools have also been refined over the past 8 years. Here, I try to compare AnyBed with the latest NET assistant tools, which includes Emulab software [61], StarBED SpringOS [62], PlanetLab User tools [63], and DETER SEER [18]. Table 7.1 shows the results of this comparison.

In terms of variation of network topology configuration supports, all the above NET assistant tools support the basic topology pattern which consists of set static routes. Only SEER supports application service settings such as HTTP server (apache [64]), DNS (bind [65]), or IRC server (ircd [66]). In the settings of layer 3 network topology, only AnyBed can offer support to create EBGP and OSPF network topologies.

Assisting in the design of network topology, Emulab software, SpringOS, and SEER equip a GUI to edit a small network topology. Although AnyBed does not support a GUI, it can create a large scale network topology in the manner that was noted in chapter 5 and chapter 6. Miwa and Hazeyama challenged a large topology emulation composed of 10,000 AS in [12, 67] with AnyBed and XENebula [68].

In terms of resource management, Emulab software covers various resources, such as the hardware of wired and wireless nodes, virtual machines, logical resources, and software for experiments. As mentioned in chapter 3.1, AnyBed's coverage of resource management is focused on logical information and software. The modularity of AnyBed leaves other resources to be managed by the assistant tool of a given NET.

In terms of the injection layer, AnyBed provides its own injection tools just as other assistant tools do. A Health check function and Logging function are offered on each of the assistant tools being compared, although each of them has its own limitation.

In the execution of scenarios, an NS-like manner can be obtained from Emulab software, SpringOS, and SEER. The scenario execution of PlanetLab user tools and of AnyBed are shell-based scripts, so users have to customize or rewrite these tools for their scenarios.

The modularity of AnyBed is superior to that of Emulab software, SpringOS, and SEER due to the layered modularity of its architecture. PlanetLab user tools may also offer modularity equal to that of AnyBed due to various tools that have been contributed and improved by PlanetLab users.

In recent years, several small testbeds have been constructed on each NET-specific assistant tool. On the contrary, AnyBed has better NET portability than the other assistant tools.

This comparison demonstrates that AnyBed is superior to the other assistant tools in creating topologies.

Table 7.1: A comparative table among NET assistant tools

Assistant Tool	Emulab Software	StarBED SpringOS	PlanetLab User Tools	DETER SEER	AnyBed	AnyBed + SpringOS
Network Topology	Basic pattern	Basic pattern	Basic pattern	With app. services	Various patterns	Various patterns
Topology Design Assistant	Small scale / Archive (GUI)	Small scale (GUI)	N/A	Small scale / Archive (GUI)	Large scale / XML (CLI)	Large scale / XML (CLI)
Resource Management	HW VM SW Logical	HW SW Logical	HW SW Logical	HW SW Logical	SW VM [1] Logical	SW VM [2] Logical
Injection SW / conf. to nodes	Frisbee	PXE-NFS DD	User tools	Frisbee	PXE+NFS / gsh tools	PXE-NFS DD gsh tools
Health Check Functions	Equipped	Equipped	Equipped	Equipped	Equipped	Equipped
Logging	Available	Available	Available	Available	Available	Available
Scenario Execution	Emulab GUI (NS-2 like)	kuroyuri (NS-2 like)	PL user tools	SEER GUI (NS-2 like)	gsh tools (partial)	kuroyuri gsh tools
Modularity	Partially	Partially	Good	Partially	Good	Good
NET Portability	Emulab like NETs	StarBED like NETs	PlanetLab like NETs	DETER like NETs	Various NETs	StarBED like NETs
Documents	Manual Website	Manual	Manual Website	Manual Website	Manual	Manual

7.2 Reproducibility of Network Emulation Experiment

Reproducibility means that an experimenter can always get the same result when he performs an experiment in the same conditions. I consider that NETs and their assistant tools require the elements described below to provide reproducibility in a network emulation experiment.

- Behavior of surrounding environment around the target
  The environment surrounding the target implementation always behaves the same when it receives the same input.

- Accurate event occurrence
  Events such as program execution must be accurately executed each time.

I consider the first element in regard to AnyBed, whose focus is on the building of experimental network topologies. Due to the consistency checker described in section 5.2.4, AnyBed can assume that the structure of topologies built from same dataset is always the same. However, if a specified topology is intrinsically unstable, AnyBed cannot assume its equivalency. Although the structure is the same, the base elements under the built topology could be different. For example, in large scale NETs, the current reserved nodes are different from the previous reserved nodes. In such cases, because the capabilities of the nodes are different from those of the previous ones, the experimental result can be different. To prevent such a difference, it would appear that after an experimenter considers the influence of various elements, he must try to use the same elements.

In regard to the second element, I consider that the scenario system in assistant tools can realize the accurate execution of sequential events. The scenario system generates events on each node in the time sequence specified by the experimenter. AnyBed is equipped with gsh tools as a scenario system. Although the tool is simple, an experimenter is able to specify the time sequence using it, which can help to create reproducibility in network emulation experiments.

7.3 Scalability in Massive Network Topology Consisting of Virtual Nodes

Recent advances in virtualization technologies have multiplied the number of nodes in NETs by several times. We therefore tested the scalability of AnyBed anticipating that a user would use it in thousands of virtual nodes. In the case of 10000 virtual nodes, AnyBed and the XENebula toolset [69] took 2.5 days to build a network topology. This result shows that the current AnyBed implementation is not scalable in a massive network topology.

With the use of virtualization technologies, I consider that it will become more common for network researchers to use AnyBed to perform experiments with more than hundreds of virtual nodes in lab-level NETs. However, in a massive environment, problems still remain, in addition to that of scalability, such as node control, measurement, and anomaly detection. These too, will be the subject of future works.

7.4 Abstraction Level and Accuracy of the Emulation

The model of AS emulation in AnyBed is simple: an inter-AS network emulated by AnyBed consists of a single AS border gateway router and a number of network links to other ASes. This simplification would make it difficult to emulate ASes connected by multiple links via multiple border gateway routers. Multiple border gateway routers on a single AS have to be emulated when a target experiment requires high fidelity emulation of network links, because they are popularly used for redundantly connecting to other ASes.

Although AnyBed can functionally generate configurations on an AS which consists of many border gateway routers, the currently used dataset does not contain such information. It only contains the relationships of one AS to another AS.

In addition, the routing policy of each border gateway router on an emulated inter-AS network might not be accurately emulated, because a route filter on each bgpd.conf which was generated by AnyBed to implement a routing policy on the border gateway router is simply inferred from the AS relationships. I think that the original RouteView dataset and the Routing Assets Database (RADb) [70] can help us to infer the

relationships between border gateway routers and their policies.

The current implementation can only emulate an inter-AS BGP topology or an OSPF network inside an AS separately. I plan to extend the implementation to construct an end-to-end network that combines the network inside an AS with the AS-level network by implementing iBGP mechanisms in AnyBed.

In terms of accuracy of emulation, there are several differences between emulated networks and real networks. AnyBed emulates the original commercial routers used by PC routers. Many recent commercial routers use a hardware engine for packet forwarding and perform forwarding at different speeds. Moreover, its implementation of routing protocols is also different from that of PC routers and would thus exhibit different behaviors. In addition, a PC router cannot emulate the vendor-specific features of commercial routers. If I use an actual commercial router as part of a NET which is adopted in Remote Network Labs [71], or if router simulators such as CISCO 7200 Simulator [72] are used, these differences can be reduced.

I plan to emulate the performance characteristics of each network link using netem [24] so the fidelity of the network link emulation will depend on the fidelity of netem. To do this, its fidelity must be estimated, and if it does not have high enough fidelity, high fidelity emulation must somehow be provided, for example, by connecting actual networks into the emulated inter-AS network to add fidelity. This would require estimating the degree of fidelity that is required in experiments on the emulated Internet.

Chapter 8

Conclusion

In this dissertation, I presented an efficient architecture that is designed to expedite the preparations involved in network emulation experiments. I consider that the time-consumption is caused by four principal problems. The first problem is that topology configuration has no reusability. The second is that there are differences in assistant tools and experimental procedures on each NET. The third problem is the lack of compatibility and complementarity among assistant tools. The last one is the difficulty in emulating a realistic topology.

In order to solve these problems, I propose AnyBed architecture in this dissertation. AnyBed architecture divides network topology information for an experiment into two parts: logical network information and physical network information. These two streams of information are combined in such a way as to build an experimental network for the preparation process which achieves topology reusability. AnyBed architecture also has a common node configuration mechanism and a layered modular architecture, which achieves system portability and compatibility with other assistant tools. This study has also explored a method to emulate the proper size of an AS-level network picked from the public dataset and a method that emulates an actual OSPF network topology that is operated on NETs.

I implemented these proposals as an AnyBed toolset, which has been released as open-source software. The evaluation results and feedback of various users has confirmed that AnyBed can expedite the preparation of users' experiments.

8.1 Contributions

To attain the goal of expediting users' experiments on NETs, a number of issues have been addressed which yielded the following contributions.

- In order to expedite experiments on NETs, I propose my model of network emulation experiments. My proposed model for network experiments enables users to perform their experiments on different NETs that are suitable for the scale of their experiments using the same assistant tools.

- On the basis of the model previously described, I proposed AnyBed architecture. To realize reusability of a network topology configuration, I divided network topology information for an experiment into two parts: logical network information and physical network information. This division achieves a reusability of network topology among NETs, which offers users the advantage of a reduced workload in such cases by reproducing the topology of a past experiment on another large scale NET, and moving that topology from real nodes to virtual nodes.

- I have also implemented the proposed architecture as an AnyBed toolset, which has been released as open-source software. This release has attracted several regular users of AnyBed, whose past experiments using it are summarized in [11]. Workshops for these users, which included an AnyBed session, were held in 2008 and 2009 and participants also implemented tools such as XENebula [12], and XBurner [13] using a number of AnyBed components.

- I designed and implemented several tools that can emulate various parts of the Internet. To emulate inter-AS BGP topology, the caida-topology-filter tool was implemented as part of the AnyBed toolset. With this tool, users can emulate the proper size of an inter-AS topology based on the actual AS-level topology of the Internet. For intra-AS OSPF topology, I implemented ONTES: an OSPF Network Topology Emulation System. ONTES infers router configurations from the OSPF Link State Database (LSDB) obtained from one actual router on a backbone network, and then emulates OSPF network topology with the inclusion of the OSPF interface cost settings and assigned IP addresses on NETs.

- To provide portability of tools, I designed and implemented an AnyBed portable toolset. Regardless of the original OSes installed in NET nodes, the master server of the toolset provides the nodes with the specified OS image and software via a PXE boot and NFS root. Users can use the same toolset while performing experiments on various NETs by accessing one master server that packages these toolsets into the NETs.

- I have proved by evaluation results that the proposed architecture of AnyBed can expedite network emulation experiments. Using AnyBed, the total time it takes to construct a complex BGP topology on 150 nodes is 113 s. This result shows that AnyBed can generate configuration files and can deploy them on a large scale NET in a short enough time period.

References

[1] Toshiyuki Miyachi, Kenichi Chinen, and Yoichi Shinoda. Automatic configuration and execution of internet experiments on an actual node-based testbed. In *Proceedings of 2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities(TridentCom)*, pp. 274–282, February 2005.

[2] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

[3] NICT Hokuriku Research Center. `http://starbed.nict.go.jp/`.

[4] T. Benzel et.al. Design, Deployment, and Use of the DETER Testbed. In *Proceedings of DETER community workshop 2007*, Aug. 2007.

[5] PLANETLAB - An open platform for developing, deploying, and accessing planetary-scale services. `http://www.planet-lab.org/`.

[6] Information Sciences Institute. NS-2 network simulator. Software Package, 2003. `http://www.isi.edu/nsnam/ns/`.

[7] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006.

[8] Shashi Phoha Glenn Carl, George Kesidis and Pennsylvania State University Bharat Madan. Preliminary BGP Multiple-Origin Autonomous Systems (MOAS)

Experiments on the DETER Testbed. In *Proceedings of the Deter Community Workshop 2006*, June 2006.

[9] J. Moy. OSPF Version 2. RFC 2328 (Standard), April 1998. Updated by RFC 5709.

[10] M. Bhatia, V. Manral, M. Fanto, R. White, M. Barnes, T. Li, and R. Atkinson. OSPFv2 HMAC-SHA Cryptographic Authentication. RFC 5709 (Proposed Standard), October 2009.

[11] Hiroaki Hazeyama, Masatoshi Enomoto, and Mio Suzuki. A Consideration about the experiments of Overlay Network Applications on Network Emulation Testbeds (in Japanese). In *IEICE-IA2009-39*, Vol. 109, pp. 37–42, September 2009.

[12] Shinsuke Miwa, Mio Suzuki, Hiroaki Hazeyama, Satoshi Uda, Toshiyuki Miyachi, Youki Kadobayashi, and Yoichi Shinoda. Experiences in emulating 10K AS topology with massive VM multiplexing. In *Proceedings of The First ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures(VISA 2009)*, August 2009.

[13] Toshiyuki Miyachi, Shinsuke Miwa, and Yoichi Shinoda. XBurner: Design of the traffic generator platform using XENebula (in Japanese). In *Proceedings of The Second Internet and Operation Technology Symposium (IOTS2009)*, Dec 2009.

[14] Toshiyuki Miyachi Shinsuke Miwa, Ken ichi Chinen, and Yoichi Shinoda. On the Nature of Network Experiments —Issues to Automate Network Experiments—. In *Proceedings of The 20th IFIP Int. Conference on Testing of Communicating Systems and the 8th Int. Workshop on Formal Approaches to Testing of Software (TESTCOM/FATES2008)*, 2008.

[15] Ssfnet: Scalable simulation framework - network models. http://www.ssfnet.org. See http://www.ssfnet.org/publications.html for their publications.

[16] NSF/DHS EMIST Project. `http://emist.ist.psu.edu/`.

[17] Stephen Schwab, Brett Wilson, Calvin Ko, and Alefiya Hussain. Seer: A security experimentation environment for deter. In *Proceedings of the Deter Community Workshop on Cyber Security Experimentation and Test 2007*, August 2007.

[18] Deterlab. The Security Experimentation EnviRonment (SEER). `http://seer.isi.deterlab.net/trac`.

[19] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. In *Technical Report UM-CSE-TR-456-02, University of Michigan*, 2002.

[20] K. Calvert and M. Doar and E. Zegura. Modeling Internet Topology. In *IEEE Transactions on Communications*, Dec. 1997.

[21] TIERS. Tiers Topology Generator. `http://www.isi.edu/nsnam/ns/ns-topogen.html#tiers`.

[22] Alberto Medina and Anukool Lakhina and Ibrahim Matta and John Byers.

[23] Lechang Cheng, Norm C. Hutchinson, and Mabo R. Ito. Realnet: A topology generator based on real internet topology. In *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops (AINAW '08)*, pp. 526–532, 2008.

[24] S. Hemminger et.al. Network Emulation with NetEm. In *Proceedings of Linux Conf Au 2005*, April 2005.

[25] Mark Carson and Darrin Santay. NIST Net - A Linux based Network Emulation Tool. In *ACM SIGCOMM Computer Communications Review*, Vol. 33, pp. 111–126, July 2003.

[26] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 1997.

[27] The iperf TCP/UDP Bandwidth Measurement Tool. `http://iperf.sourceforge.net/`.

[28] The Public Netperf Homepage. `http://www.netperf.org/netperf/`.

[29] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon: a flow-level traffic generator for router and network tests. In *Proceedings of the joint international conference on Measurement and modeling of computer systems, 2004(SIGMETRICS '04)*, pp. 392–392, 2004.

[30] M. Oe, Y. Kadobayashi, and S. Yamaguchi. An implementation of a hierarchical IP traceback architecture. In *Proceedings of IPv6 Workshop, SAINT 2003, Orland, USA*, Jan. 2003.

[31] GENI Rspec. `http://groups.geni.net/geni/attachment/wiki/GeniRspec/rspec-draft-v0.5.doc`.

[32] Intel Corporation, Preboot Execution Environment (PXE) Specifiction Version 2.1, September 1990.

[33] Junichi Uekawa. Dsh - dancer's shell / distributed shell. `http://www.netfort.gr.jp/~dancer/software/dsh.html`.

[34] J. Postel and J.K. Reynolds. Telnet Protocol Specification. RFC 854 (Standard), May 1983. Updated by RFC 5198.

[35] R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006.

[36] M. Wasserman and T. Goddard. Using the NETCONF Configuration Protocol over Secure SHell (SSH). RFC 4742 (Proposed Standard), December 2006.

[37] T. Goddard. Using NETCONF over the Simple Object Access Protocol (SOAP). RFC 4743 (Proposed Standard), December 2006.

[38] E. Lear and K. Crozier. Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP). RFC 4744 (Proposed Standard), December 2006.

[39] GNU Zebra. `http://www.zebra.org/`.

[40] Takuji Iimura, Hiroaki Hazeyama, and Youki Kadobayashi. Distributed scalable multi-player online game servers on peer-to-peer networks. *IPSJ Digital Courier*, Vol. 1, pp. 75–90, 2005.

[41] K. Masui and Y. Kadobayashi. A Role-Based Peer-to-Peer Approach to Application-Oriented Measurement Platforms. In *Proceedings of The Third Asian Internet Engineering Conference (AINTEC 2007)*, Nov. 2007.

[42] Hiroaki Hazeyama, Mio Suzuki, Shinsuke Miwa, Daisuke Miyamoto, and Youki Kadobayashi. Outfitting an Inter-AS Topology to a Network Emulation TestBed for Realistic Performance Tests of DDoS Countermeasures. In *Proceedings of Workshop on Cyber Security Experimentation and Test (CSET'08)*, August 2008.

[43] Hiroaki Hazeyama, Mio Suzuki, Shinsuke Miwa, Satoshi Uda, Toshiyuki Miyachi, Ken ichi Chinen, Youki Kadobayashi, and Youichi Shinoda. NERDBOX : An Emulated Internet for Scalability Tests of Running Codes. In *ACM SIGCOMM 2008, Demo Session*, August 2008.

[44] Internet Systems Consortium, Inc. ISC Domain Survey: Number of Internet Hosts. `http://www.isc.org/index.pl?/ops/ds/host-count-history.php`.

[45] G. Huston. The 32-bit AS Number Report. `http://www.potaroo.net/tools/asn32/`.

[46] C. Gong et.al. Single Packet IP Traceback in AS-level Partial Deployment Scenario. In *Proceedings of IEEE GLOBECOM 2005*, Nov. 2005.

[47] CAIDA: The Cooperative Association for Internet Data Analysis. The CAIDA AS Relationships Dataset. `http://www.caida.org/data/active/as-relationships/`.

[48] K. Zhang et.al. Performing BGP Experiments on a Semi-realistic Internet Testbed Environment. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, 2007.

[49] L. Subramanian et.al. Listen and Whisper: Security mechanisms for BGP. Proceeding of NSDI ' 04, Mar. 2004.

[50] R. Cherto et.al. Topology Generation, Instrumentation, and Experimental Control Tools for Emulation Testbeds. In *Proceedings of DETER community workshop 2006*, June. 2006.

[51] Cheng Jin, Qian Chen, and Sugih Jamin. Inet: Internet topology generator, 2000.

[52] University of Oregon Route Views Project. Route Views Project Page. `http://www.routeviews.org/`.

[53] X. Dimitropoulos et.al. AS Relationships: Inference and Validation. *ACM SIGCOMM Computer Communication Review (CCR)*, Vol. 37, No. 1, pp. 29–40, 2007.

[54] IPv6 Team Of State Key Laboratory of Software Development Environment,BeiHang University. IPv6 Backbone Network Topology (A Map of Ipv6 Internet). `http://ipv6.nlsde.buaa.edu.cn/index.htm`.

[55] Toshiyuki Miyachi, Kenichi Chinen, and Yoichi Shinoda. StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software. In *Proceedings of International Conference on Performance Evaluation Methodlogies and Tools (Valuetools) 2006*, October 2006.

[56] Graphviz - Graph Visualization Software. `http://www.graphviz.org/`.

[57] Robert Ricci, Chris Alfeld, and Jay Lepreau. A solver for the network testbed mapping problem. In *ACM SIGCOMM Computer Communications Review 33(2)*, 2003.

[58] Jelena Mirkovic, Brett Wilson, Alefiya Hussain, Sonia Fahmy, Peter Reiher, Roshan Thomas, and Stephen Schwab. Automating ddos experimentation. In *Proceedings of the Deter Community Workshop on Cyber Security Experimentation and Test 2007*, August 2007.

[59] Yu-Lun Huang et.al. SWOON: A Testbed for Secure Wireless Overlay Networks. In *Proceedings of the CyberSecurity Experimentation and Test (CSET) Workshop*, 2008.

[60] WIDE Project: Widely Integrated Distributed Environment. `http://www.wide.ad.jp/`.

[61] Emulab Software Distributions. `http://www.emulab.net/software.php3`.

[62] SpringOS : Network Experiment Supporting Software. `http://www.starbed.org/softwares/index.html`.

[63] PLANETLAB User Tools. `http://www.planet-lab.org/tools`.

[64] The Apache Software Foundation. Apache HTTP Server Project. `http://httpd.apache.org/`.

[65] Internet Systems Consortium. ISC BIND. `https://www.isc.org/software/bind`.

[66] irc@irc.funet.fi. IRCD. `http://www.nic.funet.fi/~irc/server/`.

[67] Hiroaki Hazeyama, Satoshi Ohta, Shinsuke Miwa, Toshiyuki Miyachi, Mio Suzuki, Satoshi Uda, Ken ichi Chinen, Masatoshi Enomoto, Youki Kadobayashi, and Youichi Shinoda. Emulating over 10K AS topology with massive VM multiplexing. In *ACM SIGCOMM 2009, Demo Session*, August 2009.

[68] XENebula. `http://tbn.starbed.org/XENebula/`.

[69] Shinsuke Miwa, Mio Suzuki, Hiroaki Hazeyama, Satoshi Uda, Toshiyuki Miyachi, Youki Kadobayashi, and Youichi Shinoda. Building mimetic internet - a trial to emulate inter-AS networks - (in japanese). In *Internet Conference 2007*, pp. 41–48, October 2007.

[70] Merit Network Inc. RADb Routing Assets Database.

[71] Huan Liu and Dan Orban. Remote Network Labs: An On-Demand Network Cloud for Configuration Testing. In *Proceedings of Workshop: Research on Enterprise Networking (WREN 2009)*, August 2009.

[72] C. Fillot. Cisco 7200 simulator. `http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator`.

[73] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: A peer-to-peer approach to scalable multi-player online games. In *Proceedings of ACM NetGames 2004*, pp. 116–120, Aug. 2004.

[74] Kenji Masui and Youki Kadobayashi. An application-oriented measurement platform built on role-based p2p network: Performance evaluation and deployment scenarios (in japanese). *Transactions of IPSJ*, Vol. 50, No. 2, pp. 709–720, 2月 2009.

[75] H. Fujiwara. xdt. `http://sourceforge.net/projects/xdt/`.

[76] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications.

In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[77] Ken Wakasa et al. Devised the placement plan to the ISP environment and a scenario plan for the traceback prior experiment on the Internet (in Japanese). In *Proceedings of Computer Security Symposium 2008(CSS2008)*, October 2008.

Appendix A

List of Publications

A.1 Journal

1–1. Mio Suzuki, Hiroaki Hazeyama, Daisuke Miyamoto, Shinsuke Miwa, and Youki Kadobayashi, "Expediting Experiments across Testbeds with AnyBed: A Testbed-Independent Topology Configuration System and Its Tool Set", *IEICE Transactions on Information and System*, Vol. E92-D, No. 10, pp. 1877-1887, October 2009.

A.2 International Conference

2–1. Mio Suzuki, Hiroaki Hazeyama, and Youki Kadobayashi, "Expediting experiments across testbeds with AnyBed: a testbed-independent topology configuration tool" In *Proceedings of 2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, March 2006.

2–2. Mio Suzuki, Teruaki Yokoyama, Takuji Iimura, Shigeru Kashihara, Takeshi Okuda, and Suguru Yamaguchi, "Nazca: A geographic location service" In *Proceedings of the 2007 International Symposium on Applications and the Internet (SAINT2007), Workshop on Ubiquitous Networking and Enablers to Context-Aware Services*, January 2007.

2–3. Shinsuke Miwa, Mio Suzuki, Hiroaki Hazeyama, Satoshi Uda, Toshiyuki Miy-

achi, Youki Kadobayashi, and Yoichi Shinoda, "Experiences in emulating 10K AS topology with massive VM multiplexing" In *Proceedings of The First ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures(VISA 2009)*, August 2009.

2–4. Hiroaki Hazeyama, Mio Suzuki, Shinsuke Miwa, Daisuke Miyamoto, and Youki Kadobayashi, "Outfitting an Inter-AS Topology to a Network Emulation TestBed for Realistic Performance Tests of DDoS Countermeasures" In *Proceedings of Workshop on Cyber Security Experimentation and Test (CSET'08)*, August 2008.

2–5. Jungsuk Song, Daisuke Inoue, Masashi Eto, Mio Suzuki, Satoshi Hayashi, and Koji Nakao, "A Methodology for Analyzing Overall Flow of Spam-Based Attacks" In *Proceedings of 16th International Conference on Neural Information Processing (ICONIP2009)*, December, 2009.

2–6. Masashi Eto, Daisuke Inoue, Mio Suzuki, and Koji Nakao, "A Statistical Packet Inspection for Extraction of Spoofed IP Packets on Darknet" In *Proceedings of The Fourth Joint Workshop on Information Security (JWIS2009)*, August 2009.

2–7. Hideo Masuda, Masahide Nakamura, Mio Suzuki and Michio Nakanishi, "Implementation and Evaluation of Authorized Access LAN Sockets Using PPPoE" In *Proceedings of The 2002 International Symposium on Applications and the Internet (SAINT2002)*, January, 2002.

A.3 Technical Report

3–1. 鈴木未央, 櫨山寛章, 榎本真俊, 三輪信介, 門林雄基, "ネットワークエミュレーションテストベッドを用いた実 OSPF トポロジ模倣システム", インターネットコンファレンス 2009 (IC2009), pp.15-24, 2009 年 10 月.

3–2. 鈴木未央, 櫨山寛章, 門林雄基, "AnyBed: クラスタ環境に依存しない実験ネットワーク構築支援機構の設計と実装" 情報処理学会研究報告, 2004-OS-96-(17), pp.121-128, 2004 年 6 月.

3–3. 鈴木未央，桝田秀夫，中西通雄, "PPPoE を用いた情報コンセント接続時の利用者認証", 2001 年 PC カンファレンス, pp.196-197, 2001 年 8 月.

3–4. 櫨山寛章，榎本真俊，鈴木未央, “ネットワークエミュレーションテストベッドを用いたオーバーレイネットワークアプリケーションの実験手法に関する一考察”, 信学技報, vol. 109, no. 208, IA2009-39, pp. 37-42, 2009 年 9 月.

3–5. 三輪信介, 鈴木未央, 櫨山寛章, 宇多仁, 宮地利幸, 門林雄基, 篠田陽一, “模倣インターネット環境の構築 — AS 間ネットワーク構築の試行 —”, インターネットコンファレンス 2007（IC2007）, pp.41-48, 2007 年 10 月.

3–6. 力武 健次, 衛藤 将史, 鈴木 未央, 井上 大介, 中尾 康二, 小林 悟史, 秋葉 澄伸, “NGN/IPv6 セキュリティ試験システムの設計と評価”, 信学技報, vol. 109, no. 86, ICSS2009-26, pp. 97-102, 2009 年 6 月.

3–7. 宮本大輔, 鈴木未央, 櫨山寛章, 門林雄基, “フィッシング攻撃対策ツールの有効性評価のためのスキャナの提案”, 暗号と情報セキュリティシンポジウム（SCIS2006）, 2006 年 1 月.

3–8. 桝田 秀夫, 大角 祐介, 鈴木 未央, 中西 通雄, “生協食堂における無線 LAN サービス実証実験”, 2002 年 PC カンファレンス, pp.348-349, 2002 年 8 月.

3–9. 桝田秀夫, 鈴木未央, 長瀧寛之, 中村聡史, 海貝明道, 小川剛史, 中西通雄, “Linux システムを中心とした授業支援・運用支援システムの構築”, 2001 年 PC カンファレンス, pp.34-35, 2001 年 8 月.

3–10. 桝田秀夫, 鈴木未央, 中西通雄, “PPPoE を用いた認証付き情報コンセントの実装と評価”, DICIMO 2001, pp. 379-384, 2001 年 6 月.

3–11. 桝田 秀夫, 鈴木 未央, 中西 通雄, “PPPoE を利用した認証付き情報コンセントの実装と評価”, 情報処理学会研究報告 分散システム/インターネット運用技術 (DSM2001-05), pp.19-24, 2001 年 5 月.

Lecture and Seminar

4–1. 鈴木未央, “汎用テストベッドトポロジ構築ツール AnyBed”, Traceable Network Developers Conference 2009 (TNDC ’ 09), 2009 年 7 月

4–2. 鈴木未央, “フル AS トポロジーエミュレーションへの取り組み”, StarBED Technical Workshop 2007, 2007 年 11 月

Demonstration and Poster Session

5–1. 鈴木未央, 櫨山寛章, 榎本真俊, 三輪信介, 門林雄基, “模倣インターネット - 実 OSPF ネットワークの模倣 -”, インターネットコンファレンス 2009 (IC2009), デモンストレーション, 2009 年 10 月

5–2. Daisuke Inoue, Mio Suzuki, Masashi Eto, Katsunari Yoshioka, and Koji Nakao, “DAEDALUS: Novel Application of Large-Scale Darknet Monitoring for Practical Protection of Live Networks” In *Proceedings of 12th International Symposium On Recent Advances in Intrusion Detection (RAID 2009)*, Poster Session, September 2009.

5–3. Hiroaki Hazeyama, Satoshi Ohta, Shinsuke Miwa, Toshiyuki Miyachi, Mio Suzuki, Satoshi Uda, Ken ichi Chinen, Masatoshi Enomoto, Youki Kadobayashi, and Youichi Shinoda, “Emulating over 10K AS topology with massive VM multiplexing” In *ACM SIGCOMM 2009*, Demo Session, August 2009.

5–4. Hiroaki Hazeyama, Mio Suzuki, Shinsuke Miwa, Satoshi Uda, Toshiyuki Miyachi, Ken-ichi Chinen, Youki Kadobayashi, and Youichi Shinoda., “NERDBOX : An Emulated Internet for Scalability Tests of Running Codes” In *ACM SIGCOMM 2008*, Demo Session, August 2008.

Appendix B

Recent Experiments Performed with AnyBed

In this chapter, I introduce recent experiments performed with AnyBed briefly.

B.1 Zone Federation Model

Iimura et al. have evaluated *libcookai* that is the implementation of *Zone Federation Model*(ZFM). ZFM enables Pure P2P Overlay Multiplayer Online Game Infrastructure. The detail of ZFM is described in his paper [40, 73].

In his first experiment, he run totally 700 processes on 7 nodes in StarBED. This experiment aimed for a measurement about propagation delay of messages among a large number of processes. Next experiment used 295 nodes and up to 1,000 processes on each node. On this experiment, the interface of each nodes is configured to have 100 ms delay by Dummynet.

B.2 N-TAP

Masui et al. have performed experiment to evaluate his implementation of *N-TAP*, that means "Pure P2P Overlay Network Measurement Infrastructure". He used two NETs such as StarBED and PlanetLab, and compare the results of the ideal environment and the realistic environment. The detail of N-TAP and his experiments is described in his paper [74].

B.3 xdt and Chord

Fujiwara and Kadobayashi have evaluated *xdt* [75], that is a library of multi-threaded message dispatcher and network dispatcher. They also developed and evaluated an implementation of Chord [76] using xdt. They used 100 nodes, and run 300 processes on each node. Their purposes are observing the whole process of overlay-network construction and measuring the time before the network was stable.

B.4 IP Traceback

Hazeyama et al. performed an experiment about his *IP Traceback* system using 200 up to 680 nodes. He picked up Japanese ASes using caida-topology-filter tool, and emulate relations of those ASes using AnyBed. On that network topology, he simulated the deployment scenario of the IP traceback system. Based on this result, he consider real deployment scenario of the system [77].

B.5 Visualizing BGP behavior

In the Cloud Computing Competition on Interop Tokyo 2009(Interop-CCC2009), Hazeyama et al. performed visualization of BGP behavior on whole router in emulated Internet. He firstly emulated Japanese AS-level BGP network on 509 XEN virtual nodes over 50 physical nodes. Then, he visualized the BGP behavior of whole networks by collecting and summarizing BGP Update messages on each router. In this experiment, he took 1.5 hours before the network was stable.

B.6 Intellisense

Also in the Interop-CCC2009, Kuromiya et al. evaluated his implementation of *Intellisense* that enables locality-aware data transfer algorithm using overlay-networks. Using AS-level BGP network described in B.5, he chose 8 ASes and connected 48 slave-nodes to the ASes. Then he evaluated the transfer algorithm by running his implementation on the slave-nodes.

Appendix C

Input and Output of AnyBed

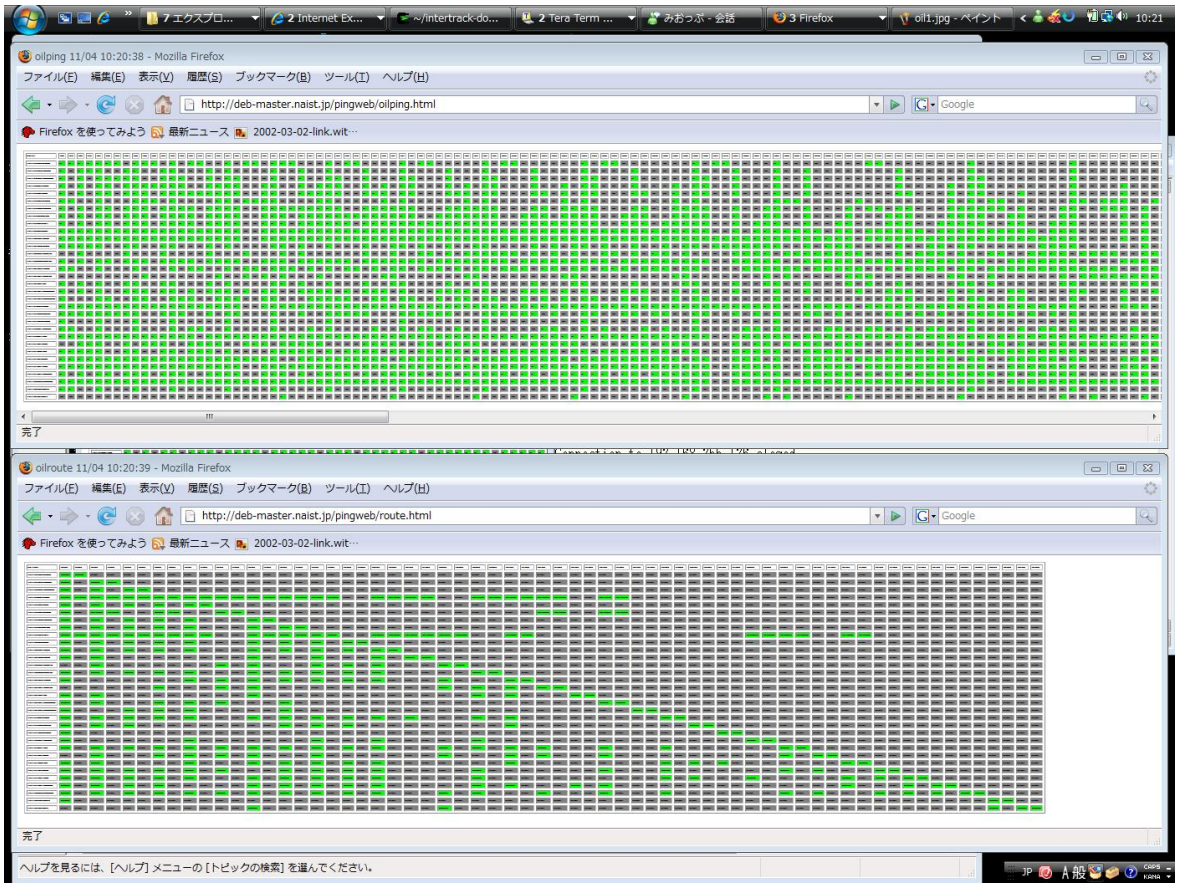

Figure C.1: Results of rearchability check and route check

```
192.168.0.1     rid-192.168.0.1-AS10010
10.15.189.2     peer-10.15.189.2-10010->9002
10.50.16.2      peer-10.50.16.2-10010->15412
10.12.223.2     peer-10.12.223.2-10010->9225
10.12.106.2     peer-10.12.106.2-10010->9370
10.2.148.2      peer-10.2.148.2-10010->4725
10.75.171.2     peer-10.75.171.2-10010->2516
10.37.183.2     peer-10.37.183.2-10010->2497
192.168.0.2     rid-192.168.0.2-AS10015
10.63.34.2      peer-10.63.34.2-10015->2516
10.11.12.2      peer-10.11.12.2-10015->17676
10.79.76.2      peer-10.79.76.2-10015->4713
10.46.194.2     peer-10.46.194.2-10015->9351
192.168.0.3     rid-192.168.0.3-AS10021
10.52.204.2     peer-10.52.204.2-10021->2914
10.33.230.2     peer-10.33.230.2-10021->15412
10.23.23.2      peer-10.23.23.2-10021->3738
10.60.89.2      peer-10.60.89.2-10021->2516
10.22.82.2      peer-10.22.82.2-10021->2497
10.12.172.2     peer-10.12.172.2-10021->6303
...(snip)...
10.46.37.1      peer-10.46.37.1-9950->17841
192.168.9.212   rid-192.168.9.212-AS9957
10.54.122.1     peer-10.54.122.1-9957->15412
10.0.136.1      peer-10.0.136.1-9957->18302
10.27.175.1     peer-10.27.175.1-9957->10036
10.37.178.1     peer-10.37.178.1-9957->9318
10.41.195.1     peer-10.41.195.1-9957->38091
10.66.33.1      peer-10.66.33.1-9957->10026
192.168.9.213   rid-192.168.9.213-AS9989
10.63.43.1      peer-10.63.43.1-9989->4637
10.34.145.1     peer-10.34.145.1-9989->4657
10.57.42.1      peer-10.57.42.1-9989->13276
10.13.177.1     peer-10.13.177.1-9989->7786
10.65.21.1      peer-10.65.21.1-9989->5511
192.168.9.214   rid-192.168.9.214-AS9993
10.77.110.1     peer-10.77.110.1-9993->2914
10.55.162.1     peer-10.55.162.1-9993->15412
10.35.51.1      peer-10.35.51.1-9993->9225
10.5.42.1       peer-10.5.42.1-9993->2516
10.14.44.1      peer-10.14.44.1-9993->2497
```

Figure C.2: pingman.conf

```
10.0.1.0/24
10.0.10.0/24
10.0.115.0/24
10.0.12.0/24
10.0.130.0/24
10.0.135.0/24
10.0.139.0/24
10.0.153.0/24
10.0.163.0/24
10.0.182.0/24
...(snip)...
10.9.242.0/24
10.9.28.0/24
10.9.3.0/24
10.9.38.0/24
10.9.6.0/24
10.9.61.0/24
10.9.66.0/24
10.9.67.0/24
10.9.74.0/24
10.9.83.0/24
```

Figure C.3: routecheck.conf

```
::1         localhost localhost.my.domain
127.0.0.1         localhost localhost.my.domain

#         myrid-Int-myip-peerip
192.168.0.1     192.168.0.1-AS10010     xen0001
10.15.189.1     192.168.0.1-Int-AS10010-AS9002
10.50.16.1      192.168.0.1-Int-AS10010-AS15412
10.12.223.1     192.168.0.1-Int-AS10010-AS9225
10.12.106.1     192.168.0.1-Int-AS10010-AS9370
10.2.148.1      192.168.0.1-Int-AS10010-AS4725
...(snip)....
10.13.177.2     192.168.9.213-Int-AS9989-AS7786
10.65.21.2      192.168.9.213-Int-AS9989-AS5511
10.84.60.1      192.168.9.213-Int-AS9989
192.168.9.214   192.168.9.214-AS9993    xen2500
10.77.110.2     192.168.9.214-Int-AS9993-AS2914
10.55.162.2     192.168.9.214-Int-AS9993-AS15412
10.35.51.2      192.168.9.214-Int-AS9993-AS9225
10.5.42.2       192.168.9.214-Int-AS9993-AS2516
10.14.44.2      192.168.9.214-Int-AS9993-AS2497
10.41.130.1     192.168.9.214-Int-AS9993
```

Figure C.4: /etc/hosts file generated by AnyBed