Fault Injection on a Large-Scale Network Testbed

Toshiyuki Miyachi
NICT[†]
4-2-1 Nukui-Kitamachi
Koganei, Tokyo, Japan
miyachi@nict.go.jp

Razvan Beuran
NICT[†]
4-2-1 Nukui-Kitamachi
Koganei, Tokyo, Japan
razvan@nict.go.jp

Shinsuke Miwa
NICT[†]
4-2-1 Nukui-Kitamachi
Koganei, Tokyo, Japan
danna@nict.go.jp

Yoshiki Makino
NICT[†]
4-2-1 Nukui-Kitamachi
Koganei, Tokyo, Japan
ymakino@nict.go.jp

Satoshi Uda
JAIST[§]
1-1 Asahidai, Nomi
Ishikawa, Japan
zin@jaist.ac.jp

Yasuo Tan
JAIST[§]
1-1 Asahidai, Nomi
Ishikawa, Japan
ytan@jaist.ac.jp

ABSTRACT

In the real Internet, various types of problems are encountered. Before integrating new technologies into the Internet, developers should understand the behavior of these technologies in the event of fault occurrence. Researchers conduct experiments on network testbeds to evaluate new technologies, but the main focus of the evaluations done so far has been to study the technologies' behavior in healthy situations.

Currently, there are many network testbeds available. We developed and currently operate StarBED, an actual node-based network testbed. In this paper, we discuss methodologies to introduce faults into an experimental environment using an actual node-based network testbed. Then, we show two case studies of fault injection into our experimental environments on StarBED.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: [Network management,Network monitoring]

General Terms

Experimentation

[†]National Institute of Information and Communications Technology
[§]Japan Advanced Institute of Science and Technology

AINTEC'11, November 9–11, 2011, Bangkok, Thailand.

Keywords

Fault Injection, Network Testbed

1. INTRODUCTION

Network technologies have been merged into many elements of our daily lives, and there are many important services on the Internet. As the Internet grows, many new services and improvements for the Internet are being proposed, one after another. In order to keep these services and the Internet itself healthy, evaluations of the new proposals are important.

Evaluations of network technologies are conducted utilizing several methodologies, including various simulations and emulations. However, in many cases, the main focus of these evaluations so far has been to study the behavior of target technologies in healthy situations. Evaluating the scalability or performance of the target technologies themselves is, of course, important, but studying their behavior in critical situations is also important for understanding problems with surrounding services and the target technology itself.

We have three motivations for injecting faults into experimental environments:

- The first motivation is that we want to construct realistic experimental environments. StarBED[6][8] is a network testbed based on actual nodes. It has over 1,000 PC nodes and network equipments devoted only to network experiments. The purpose of StarBED is to evaluate implemented software and hardware in real environments. Trouble in a real environment is not rare, so it is necessary to inject faults to realistically simulate or emulate the Internet.
- The second motivation is that we would like to study the behaviors of target technologies or environments in critical or emergent situations. Experiences in recovering new technologies in trouble is indispensable for enabling quick recovery in a real situation. Thus, study-

ing how experimental environments will break in critical or emergent situations is useful for solving problems in a real environment. These information is also useful for revise their specifications or implementations.

- The third motivation arises from a characteristic of experimental environments. In evaluation environments that are isolated from real environments, experimenters can measure all of a technology's elements and do anything to them. This means that we can introduce intentional problems in these environments and measure all of their influences on the technology's operations. This is not allowed in a real environment.

In this paper, we discuss network trouble in the Internet and propose ways to reproduce situations that include these problems, especially in actual node-based network testbed. Then, we show two case studies of fault injection on StarBED.

2. NETWORK FAULTS

In this section, we describe the target network faults in this work and their characteristics. We premise target technology evaluations that can be performed in a small experimental environment are already finished; StarBED is for evaluating a technology's behavior on a large-scale network. Moreover, we do not focus on compile-time injection, because our policy on StarBED is to evaluate a running implementation in a real environment.

2.1 Areas of Failure

Network faults occur in four areas in physical piece of equipments as shown in Table 1. These faults can occur in their implementation and configuration parts. The implementation includes both software and hardware. Errors in physical equipment are caused by unexpected physical effects. Specification errors come from design bugs including mistakes in understanding the target environment, scale, and so on. Implementation errors came from bugs in implementing the standard in hardware or software. Configuration errors are problems that are not in the implementation itself. The methods of evaluating these errors should be different.

Table 1: Area in Which Fault Occur

Area of Failure	Description
Physical equipments	Hardware errors in PCs, network equipment, and cables
Specifications	Hardware and software bugs in the standards in large-scale environments
Implementations	Hardware and software bugs in implementations in large-scale environments
Configurations	Hardware and software configuration errors in large-scale environments

Table 2: Failures in TCP/IP Protocol Layers

Layer	Failure Examples
Physical and Data Link	Damaged cables Loose terminal connections Inappropriate light strength (fiber) Incorrect cable connections Broken network interfaces Connection loops Bugs in specifications or implementations Disconnected VLAN Interference (wireless)
Network	Missing route Route flap IP address misconfigurations Firewall misconfigurations Bug in specifications or implementations
Transport	Bugs in specifications or implementations Misconfigurations of protocol
Application	Misconfigurations of applications Bugs in specifications or implementations

2.2 Faults in Each Protocol Layer

In the previous section, we described the areas in which faults occur. These faults can be located in all of the protocol layers, and the technologies' behavior should be different depending on layer in which the faults occur. Table 2 lists faults in each layer. The role of the protocol is to provide connections in each layer, so when the protocol has trouble the connection quality is degraded.

When a server application dies, only its service will become unavailable, and other services on the physical machine will remain available. When a machine has a problem in its protocol stack, elements in lower layers can communicate but those in upper layers cannot communicate correctly. All of under-layer faults of a layer will be shown as almost the same symptom. The roles or functions of application software above the application layer are variable, so the symptoms are also different depending on the software purposes.

2.3 Locations of Faults

Problems occurring on end nodes are described in section 2.2. However, faults can occur everywhere in the network, and the fault should be shown differently according to the observer's location. This section describes the relation between where faults occur and where the observers are.

When an end node has problems, other nodes recognize that the node has trouble but when there is trouble in network equipment, including routers and switches, the symptoms should be different depending on the troubled equipment's peers.

Figure 1 shows an example topology. If a critical problem

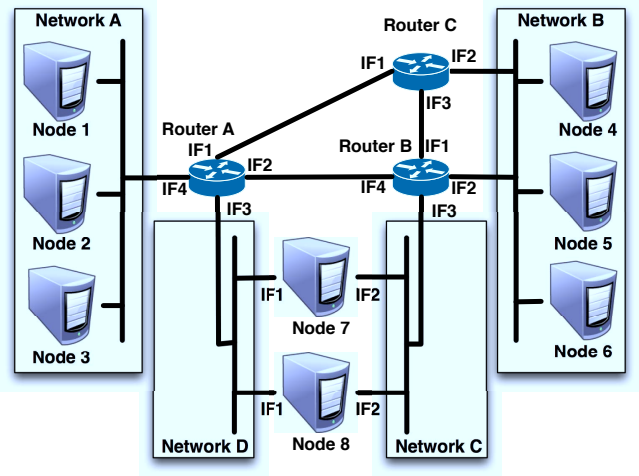

Figure 1: Example Topology

occurs on IF 4 of Router A, the whole network is divided into two parts, and nodes can only establish communication within each network. In this case, nodes in Network A think that Router A has problems, but other nodes recognize that at least IF 1, IF 2 and IF 3 of Router A are healthy.

The behavior of the routers should be change depending on the trouble caused by the route changes. Nodes in Network B might not realize that there is a problem on Router B or C if a new route exchange among the troubled routers is finished quickly and the route configurations on these nodes are appropriate excepting small fluctuations in the netwokr characteristics. On the other hand, in Networks A, C, and D, nodes will not connect to nodes in other networks when their external router has a problem.

For nodes that have multiple network interfaces, trouble on a single network interface causes complicated phenomena, as would be the case if trouble occurred on one of the network interfaces on Network C or D. In particular, in the layers above the Network Layer, the source address of the network layer should be determined by protocol in the network layer, and an available interface will be selected automatically. Thus, although it would appear that there is no problem from upper layers, there will surely be problems from the lower layers.

2.4 Fault Chains

Network faults often lead to other faults on other services. Of course, the lower layer's faults should cause upper layer faults. In the same layer, we often see elements scrambling for resources, such as the CPU clock, memory capacity, network bandwidth, and so on. In this case, a service requires a large amount of resources, and it causes faults in other services because of resource shortages.

Moreover, there are important services that are used implicitly. The Domain Name Service (DNS) is a major example of these kinds of services. When the DNS is not well, users cannot connect to other nodes using a fully qualified domain name (FQDN), which causes users to be unable to see Web pages using FQDN URLs and unable to send e-mail because the mail exchanger (MX) server of each domain cannot be found.

3. FAULTS INJECTION INTO EXPERIMENTAL ENVIRONMENTS

We described the characteristics of faults in network technologies in section 2, as well as how the effects of the faults appear as various symptoms.

In this section, we discuss the methodology to inject faults into our experimental environments.

3.1 Symptoms of Network Faults

The final circumstances of network faults are various.

Faults below the Transport Layer might appear as a link going down or as degradations of the connection quality, including an increase in the packet loss rate, an unstable round trip time (RTT), a large jitter, an unstable time to live (TTL) value, lower bandwidth, packet duplication, or a complex combination of these phenomena.

However, the processes by which the trouble occurs are different even if the ultimate consequence is the same. For example, when a network has a loop and there is no management function, such as a loop detection, the traffic goes around the loop. In this situation, the traffic volume would increase until many packets passing through the links that compose the loop are lost. In this process, when the traffic volume is lower than the link bandwidth, there should be no problem, and problems only emerge when the traffic volume exceed the limit. Now, many organizations have introduced intelligent switching hubs into their network. Thus, when a link loop is created by operational errors, only broadcast data will be amplified at first and later, all data should be amplified because entries in the forwarding database (FDB) will exceed the limit. Several stages with different symptoms that occur before the final symptom, as well as recovery stages, are observed.

Faults in applications should also be different depending on the purpose and structure of the target technologies.

3.2 Actual and Reproduced Elements

Trouble situations caused by problems in the layers below the Application Layer might be reproduced by using network emulators such as dummynet[11], netem[1] and NIST-Net[4]. These tools enable us to introduce network characteristics on a network interface. Another approach is to introduce real trouble into target environments intentionally.

For accurate results, the best method to reproduce the target problems might be the latter one. However, in many cases, it is difficult to create a realistic environment for reproducing trouble situations, and the former method is useful for avoiding these kinds of difficulties. However, when using network emulators, users should create models that

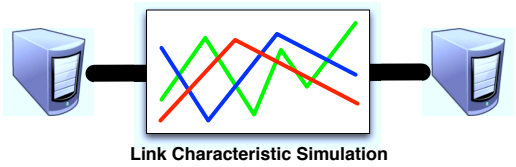

Figure 2: Network Reproduction Using Link Emulator

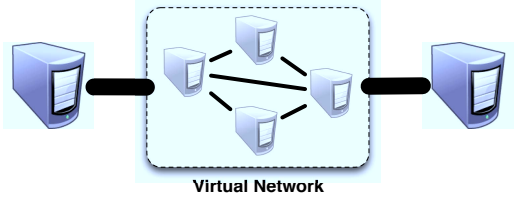

Figure 3: Network Reproduction Using Virtual Network

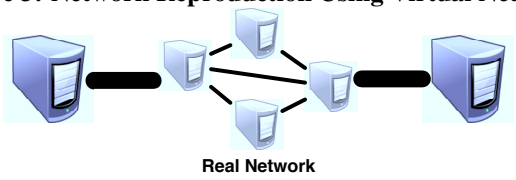

Figure 4: Network Reproduction Using Real Network

describe the processes of the trouble. Moreover, network emulators can change the link characteristics in response to users' triggers. Thus, the network emulator should change drastically according to different user configurations, and it is not suitable for reproducing the process of how different problems produce their various consequences. To reproduce the trouble situation accurately, it is necessary to model the problems in a continuous process. These kinds of modeling are not easy in real reproductions. It is also difficult to emulate physical failure in the real world, which would require users to break hardware equipments intentionally, and even then the hardware might not show the required symptom. Therefore, both methods are needed to reproduce various faults. Table 3 shows trouble symptoms and the methods that can be used to reproduce them.

There are three types of methods for imitating network trouble situations. Figures 2, 3, and 4 show these types. Figure 2 indicates in-the-middle simulation of the whole network. This type of method does not require a large amount of resources for building the environment. Figure 4 shows the method using a real environment for the whole network. It enables users to have a realistic environment for conducting experiments. However, the costs of building and controlling these environment is larger than that of simulation methods.

Figure 3 shows model that is a hybrid of those in Figure 2 and 4. As virtualization technologies are developed and used widely, we can introduce them to our environment. In this case, physical machines are emulated, but we can use actual software, including OSs, on them. Using this model,

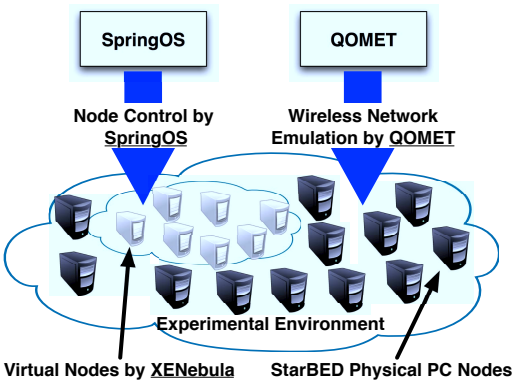

Figure 5: Our Technologies for Building Experimental Environments for Failure Situations

the cost of building environments can be reduced from that of using physical machines for all nodes.

3.3 Our Approach

We adopt an approach that is a hybrid of those in Figures 2, 3, and 4, and introduce all these types to appropriate parts in our environment.

We have tools for realizing a large-scale and realistic experimental environment on StarBED. SpringOS[6][8] is supporting software for conducting experiments. The major functions of SpringOS are resource management, power management of nodes, software installation, network topology construction, scenario driving, and so on. It facilitates us the creation of experiments. XENebula[5] enables users to run many virtual machines. The roles of XENebula are creating disk images and configurations for all virtual machines based on templates, allocating virtual machines to available physical nodes based on the computing resources of the physical machines, and running configured virtual machines on physical machines. QOMET[2] can emulate characteristics of a wireless network on a wired network. It has behavior models of wireless nodes, and it configures the wired link characteristics using network emulators.

StarBED has many PC nodes, and XENebula can amplify the scale of StarBED. SpringOS and XENebula build experimental topologies using physical and virtual machines. SpringOS can also be used for scenario control on the physical and virtual machines. QOMET is a tool for emulating wireless network behavior, but it can also be used to emulate fault situations. QOMET models wireless networks, and we can replace that model with that for fault reproduction. Figure 5 indicates the use of our technologies for fault reproduction.

4. CASE STUDIES

We executed two experiments using fault injection. The first one involved emulating home networks, power management by a home energy management system (HEMS), and

Table 3: Trouble Symptoms of Each Layer and Usable Methods

Layer	Trouble Symptoms	Usable Method
Physical and Data Link	Link goes down or connection of qualities degrades, No reachability	Real reproduction cannot be used, and network emulators should be adopted
Network	No reachability or degradation of connection quality	Network emulators can be adopted, but the trouble's indiscrete processes and in many cases it is too complex to model its; behavior; actual reproduction should be used
Transport	No reachability or degradation of connection quality	Network emulators should cover the symptoms.
Application	Service unavailable, depending on the application	Network emulation and actual reproductions should be used depending on the target services

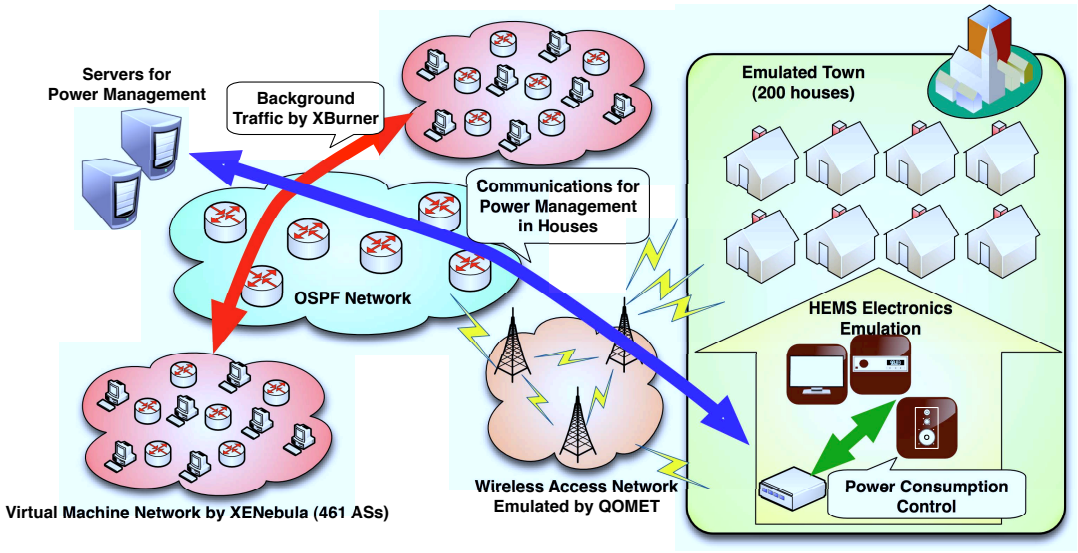

Figure 6: Topology for HEMS Emulation

wide area networks. The second one involved an environment for Cloud Computing Competition. In this section, we describe these experiments.

4.1 HEMS Emulation with Actual Network Elements

The topology of this experiment is shown in Figure 6. This experimental network has about 1100 nodes as experimental elements utilizing about 60 physical nodes. The main elements of this experiment are emulated smart houses that have HEMS electronics and servers for power management. The HEMS electronics managers send power consumption information to the servers. The servers send a message to reduce the consumption when they detect a house using too much electricity. The access network in these houses is a wireless network, and the external gateway of the network connects to an OSPF network in an emulated ISP. The servers for power management are also connected to the ISP network. The ISP has links to other ISPs that use the BGP protocol.

The smart houses with HEMS electronics are simulated according to an electric consumption models that reflects residents' behaviors. There are 200 simulated nodes in this environment executed using 20 physical nodes. The wireless access network field is reproduced by QOMET using a single physical host. The power management servers run on actual PC nodes. We built the OSPF network using 20 physical nodes that have 4 core routers and configured Zebra OSPF daemon runs on each physical nodes. The OSPF routers exchange route information to maintain reachability for the smart houses and HEMS power managers. There are two virtual machine networks created and run by XENebula, which emulate a BGP network of Japanese ISP and their neighbors. An ISP is emulated as a BGP router on a virtual machine by utilizing software router daemons, which actually exchange their routes. Each network has 461 emulated ISPs. XBurner[7] is a platform for generating traffic using XENebula and SpringOS. It controls actual applications on each virtual machine, such as *apache* HTTP daemons and *wget* HTTP clients.

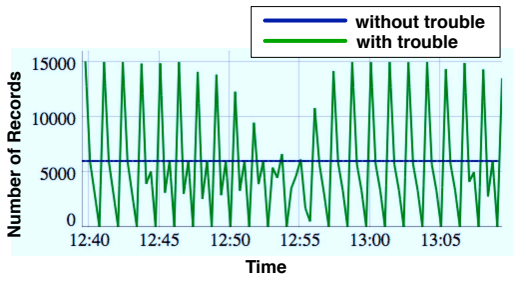

Figure 7: Instantaneous Received by Power Managers

We introduced three types of faults in this environment:

1. Changes in the link characteristics on the wireless access network
2. Misconfiguration of an OSPF router
3. Massive traffic across the OSPF network

The first fault is realized using QOMET. In this case, we assume that a thunderstorm occurred near the access network and negatively affected the wireless links. The concrete symptom of this effect is that the wireless links go down. When the thunderstorm is big enough to affect the whole area or it have a direct effect on the external gateway, none of smart houses can access the servers. Only some of the houses are unable to access the servers when the negative effect covers only a part of the area.

The second fault involved the OSPF daemon down state. Two out of four core routers have up and down states. There are four patterns: two in which one of them is downed, one where both of them are down, and one where both of them are up. These states are rotated every 20 s. This causes route recalculation, and some packets may be lost or retransmitted when the route is changes.

The last fault involves burdening the OSPF network with a massive amount of traffic using XBurner. We run *apache* and *wget* on virtual machines using XENebula and these applications generate massive HTTP traffic on the OSPF network.

We actually built two environments: one with the three kinds of faults and one without faults. The three types of faults caused communication failures between the HEMS servers and clients, and we observed the effects of these failures.

Figure 7 shows the number of records from smart houses that are received by the power management servers. Records are received at a very stable rate when there are no faults, because the houses send their data periodically. On the other hand, the rate in the presence of faults fluctuates, because the records are transmitted unevenly as TCP retransmission algorithms recover packets when they are lost. Thus, the rate at which the servers receive the records describing the electric consumption in each house is not stable. Long-lasting faults can cause the retransmission algorithms to fail.

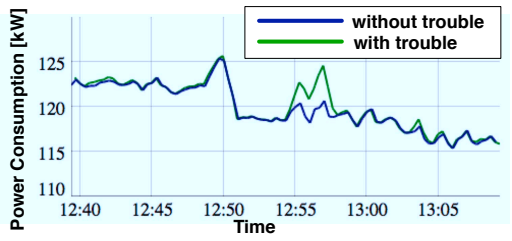

Figure 8: Power Consumption in Smart Houses

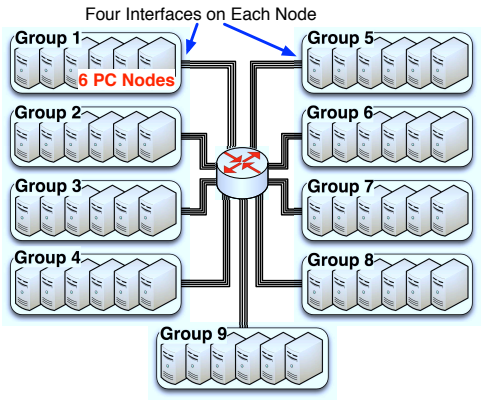

Figure 9: Topology for Cloud Computing Competition

Figure 8 shows the instantaneous power consumption in these environments. At around 12:56, power consumption in the environment with faults is larger than that without faults, because the management servers could not collect data from the houses or because the control messages from them to the HEMS clients were lost. The large loss of records is apparent in Figure 7 at around 12:55.

4.2 Cloud Computing Competition

The Cloud Computing Competition is conducted as a joint event with Interop Tokyo. One of the purposes of this event is to provide environments for participants, who can freely apply to participate in this event, to develop and apply novel and effective technologies. The committee grants awards for superior technologies in order to encourage the researchers and developers and recognize outstanding technologies. In the 2011 event, we introduced functions for introducing faults into an environment. Maintaining healthy behavior in the case of failures in the environment is especially important for cloud computing technologies. These functions for introducing faults are useful tools for ensuring the stability of proposed technologies.

The environment for this event was built on StarBED, and we made five sets of isolated environments for each participant this year. Figure 9 shows the topology for each participant.

There were 54 PCs, from which we made nine groups

with six nodes each. The PC nodes had four network interfaces: IF1 to IF4. The L2 networks were isolated; IF1 of each group composed a VLAN network, and interfaces 2, 3, and 4 also each composed a VLAN network. Thus, there were 36 L2 networks isolated by VLANs. Packets from each network were routed by a central router to other networks.

We, the provider of the environment, were actually not supposed to know the proposed technologies' behaviors in detail. Therefore, it was not clear what kinds of failures would have what kinds of impacts. Thus, we built functions to introduce failures when triggered by the participants.

We created two types failures:

- Changes in the characteristics of each network interface on each PC node
- Shutdown of the network interface of the central router

Communication quality degradation on the PC nodes was shown as software or hardware errors on the PC itself, and that on the router interface was shown as network trouble from other nodes or networks. The former type of degradation should reproduce the death of application software, and the latter one should reproduce the behavior of a denial-of-service (DoS) attack on the networks for example.

To realize these characteristic changes, we changed the network interface media. The default configuration of the network interfaces on one of the PC nodes was 1000 Mbps / Full Duplex. The link characteristics were changed by configuring the media to lower configurations, such as 10 Mbps / Half Duplex or a shutdown configuration.

Participants demonstrated their technologies for the judges and some of them used these functions to show that their products could perform effective recovery from faults. One of the products described new peer-to-peer technologies, and when some of nodes were downed, stored data in these nodes were moved to other nodes.

5. DISCUSSION

In this paper, we propose a framework for injecting faults into an experimental environment on a network testbed. However, we must address several additional issues to facilitate the execution of complete experiments involving network trouble.

First we used a wireless behavior model to inject negative effects cased by thunderstorm into a wireless access network. The model is bundled with QOMET, and it is not a failure model, although of course we can use it as a kind of failure model. In order to create various failure situations, we need more models. Such models should include the topologies and working environments of typical services. It is difficult to prepare many kinds of complete experimental environments and scenarios, because massive variations are expected. We might be able to define typical topology sets that can be connected by the experimenter to build their appropriate environment. SpringOS can control network elements on nodes, it means we can introduce our scenario of network trouble. Providing typical trouble scenario is also important factor to make realistic experiments.

Another problem is how to ensure the accuracy of fault injection. There are three kinds of accuracy. One is the correctness of the failure model, another is the precision of the controlled network elements in the experimental environments, and the third is realism of the injected failures. The first type of accuracy address how to produce clear network element behaviors and how to clearly reproduce their influences. The second one is influenced by the operational accuracy. The third comes from the complex combined behavior of the previous two. We have to guarantee these three types of accuracy.

In order to evaluate the behaviors of target technologies in the presence of network failures, we have to build infrastructure for observing elements in the experimental environments. Although we can access all resources in an experimental environment, it is difficult to know all events that happen therein.

Moreover, to understand effects caused by failures efficiently, kinds of integrated observation should be provided. Although the symptoms are different depending to faults, target technology, and so forth, but some these for general elements can be provided.

To solve all of these hurdles, we intend to create a framework that includes a complete user interface and precise models for many kinds of trouble in order to facilitate the use of our approach and making sure justification of executed experiments.

6. RELATED WORK

There are various types of fault injectors for actual node-based environments.

We note that there are many proposals for injecting faults using physical effects such as pin-level injections, electromagnetic interference (EMI), heavy-ion radiation, and so on. We do not focus these kinds of approaches because of two reasons: our targets are existing network testbeds that do not have special equipment for these effects, and these effects may cause destruction of the testbed hardware.

Network emulators such as *dummynet* and *netem* are designed to control various link characteristics, including the packet drop rate, delay, jitter and so forth. Flexlab[10] introduces link characteristics measured on Planetlab[9] into experimental environments on Emulab[12]. Modelnet can also emulate network states using network emulators on the core nodes of its environments. Network emulators enable this function.

Traffic generators are also used as fault injectors. Several traffic generators enable users to configure the packet construction and order, even if they are not correct. Moreover, massive traffic from a traffic generator can put stress on network elements. Some kinds of traffic generators just reproduce the measured traffic, and some provide interactive communications with target systems. Generators of the

former type require senders and only send traffic according to their table and receivers that only receive and discard the received packets. On the other hand, the other type of generator can send packets to any target server, and it can change its behavior according to the response of the server.

Orchestra[3] is a fault injector that inserts a fault injection layer between the target protocol layer and the lower layer and manipulates messages throw this border. However, its users must implement the inserted layer for their target protocol.

There are no model-based fault-injecting systems for a network testbed like our approach.

7. CONCLUSION

Experiments are important for ensuring the correct behavior of new network technologies before they are introduced into a real environment. Since many types of trouble are encountered in the real environments, evaluation of the behavior of new technologies in a trouble situation is required as a part of these experiments.

We propose a fault injection system for evaluating running software source code and evaluating implementation of hardware itself that utilizes our experiences in performing many kinds of network experiments on StarBED. Our tools for experimental execution, SpringOS, XENebula, QOMET, and XBurner, are also usable for fault injection.

We built a large-scale experimental environment of physical and virtual nodes using SpringOS and XENebula on StarBED. We reproduced fault situations using QOMET by modeling wireless network characteristics and applying them using network emulators. SpringOS can control the experimental scenario, including failures, and XBurner is able to generate massive traffic based on actual applications.

Our approach was used and validated in two case studies: HEMS emulation with actual network element and the Cloud Computing Competition. In the HEMS emulation, we introduced failures using actual misconfigurations and behavior modeling. The Cloud Computing Competition showed that instantaneously changing network characteristics could also reproduce some kinds of network trouble.

8. ACKNOWLEDGMENTS

We would like to thank Junya Nakata, Takashi Okada, and Yasuhiro Ohara, for providing the building parts used in our HEMS emulation experimental environment.

9. ADDITIONAL AUTHORS

Additional authors: Yoichi Shinoda (JAIST, 1-1 Asahidai, Nomi, Ishikawa, Japan, email: `shinoda@jaist.ac.jp`).

10. REFERENCES

[1] *netem — The Linux Foundation*. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.

[2] R. Beuran, J. Nakata, T. Okada, L. T. Nguyen, Y. Tan, and Y. Shinoda. A multi-purpose wireless network emulator: Qomet. In *22nd IEEE International Conference on Advanced Information Network ing and Applications (AINA 2008) Workshops, FINA 2008 symposium*, Mar. 2008.

[3] S. Dawson, F. Jahanian, and T. Mitton. Orchestra: A Fault Injection Environment for Distributed Systems. In *Proceedings of 26th International Symposium on Fault-Tolerant Computing (FTCS)*, June 1996.

[4] N. I. T. Group. *NIST Net network emulation package*. http://www-x.antd.nist.gov/nistnet/.

[5] S. Miwa, M. Suzuki, H. Hazeyama, S. Uda, T. Miyachi, Y. Kadobayashi, and Y. Shinoda. Experiences in Emulating 10K AS Topology with Massive VM Multiplexing. In *The First ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures (VISA'09)*, Aug. 2009.

[6] T. Miyachi, K. Chinen, and Y. Shinoda. StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software. In *International Conference on Performance Evaluation Methodlogies and Tools (Valuetools 2006)*, Oct. 2006.

[7] T. Miyachi, S. Miwa, and Y. Shinoda. XBurner: A XENebula-based Native Traffic-generation Platform. In *6th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom2010), Poster session*, May 2010.

[8] T. Miyachi, T. Nakagawa, K. ichi Chinen, S. Miwa, and Y. Shinoda. StarBED and SpringOS Architectures and their Performance. In *International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2011)*, Apr. 2011.

[9] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *HotNets-I '02*, Oct. 2002.

[10] R. Ricci, J. Duerig, P. Sanaga, D. Gebhardt, M. Hibler, K. Atkinson, J. Zhang, S. Kasera, and J. Lepreau. The Flexlab approach to realistic evaluation of networked systems. In *Proceedings of the Fourth Symposium on Networked Systems Design and Implementation (NSDI 2007)*, pages 201–214, Cambridge, MA, Apr. 2007.

[11] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[12] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. pages 255–270. USENIXASSOC, Dec. 2002.