Common Code Architecture for Future Internet Researches in Wireless Mesh Networks

Junhee Lee, Jinsung Lee, Sachin L. Shrestha, and Song Chong
School of Electrical Engineering and Computer Science, KAIST Daejeon, 305-701, Korea
{junhee,ljs,sachinls}@netsys.kaist.ac.kr, song@ee.kaist.ac.kr

One of new trends in future Internet research is to focus on clean-slate design. Clean-slate design starts from clean-slate thinking and verifies its new idea using simulation stuffs. Then the idea can be extended to the real network testbed. In this paper, we focus on proposing new noble architecture called *common code architecture.* It minimizes the gap between initial implementation on the simulation environment and extended implementation on the real network testbed node.

1. Introduction

The Internet has been evolved using an incremental approach in the past 30 years. We can say that it has been very successful technical evolution so far. But we are still facing set of challenges like security, scalability, quality of service and economics. Most of all, mobile users want to access the Internet while they are moving at anywhere. Thus supporting mobile Internet environment is the one of hot issues of current Internet researches. Unfortunately the basic design concept of Internet does not consider mobile environment. In future, we will have much more mobile devices that generate Internet traffics than current Internet devices. Thus it is very nature that the demand of new Internet architecture is coming.

One of major new trends in future Internet research is to focus on clean-slate design [1]. The basic concept of clean-slate design comes from designing totally new architecture from scratch. The clean-slate design starts from clean-slate thinking. Then we want to evaluate clean-slate thinking using paperwork or simulation stuffs. Construction of a new idea on the simulation environment is the best way to justify a new idea. After that, it will be verified on experimental facility and then extended to the testbed. Without verifying of a new idea that has not yet been tried under realistic conditions, it is almost impossible to guarantee a new idea working well in the real network environment.

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R01-2006-000-10753-0)

It may be inevitable performance gap between theoretical optimal solution and its first implementation on the network simulator. Theoretical approach starts with the unlimited assumption. For the example, we can assume that every node knows neighbors' status and information in the theoretical approaches. But we cannot deal with that way in the implementation on network simulator. We must exclude these assumptions and show how to exchange node's information using implementable manners. However, we can avoid the protocol performance gap when we transfer the implemented protocol from the network simulator to the real system like emulator and real-world testbed if we choose suitable implementation methodology. But it is not the common sense due to the lack of a real system knowledge and complexity of system programming. Generally we are facing difficulty of implementation on the real system even though we use very well-designed testbed. Moreover if we use general purpose platform like Linux system, we must suffer from implementation issues very far from implementation on network simulator.

In this paper, we focus on how to reduce this gap which is protocol performance difference between clean-slate implementation in a software simulation platform and a realistic network testbed node. We propose a new noble *common code architecture* that is working in both a network simulator and a real network testbed node.

The rest of this paper is organized as follows. In section 2, we investigate related work and describe what we propose for the future Internet researches in wireless mesh network in Section 3. Finally, we summarize our conclusions in Section 4.

2. Related Work

2.1 Simulator/Emulator/Testbed

A variety of network simulators, emulators, and real-world testbeds have been implemented and established so far [2]. Network simulators are the imitation of some real thing, state of affairs, or process serialized network event using simulated clock. NS-2 [12] is the most popular discrete network simulator in the research domain. GloMoSim [13] is also the well-known network simulator in the wireless domain. These simulators from research domain could be extended to commercial network simulators [16, 17]. An emulator duplicates functions of one system using a different system. Generally emulator uses a real system clock not a simulation clock. It can handle real things like real application workload or real physical behaviors and can deal with concurrent events at the same time using different way from a simulator. Network emulators offer also a viable alternative to live experimental networks [3, 9, 10]. Unlike previous emulator approaches, [4] utilizes DSP/FPGA for digital emulation of signal propagation. The testbed is a platform for experimentation for large development projects. With diverse purposes, many testbeds have been established such as [5, 6, 18, 19].

We focus on here how to reduce protocol performance gap of implementations in the simulator and the real system node. A Click modular router [11] is one of solution for that purpose. It is good software toolkit to implement what they have clean-slate attempt in the router. [7] is very similar work with our approach. They can share protocol stack codes implemented by Java in simulation, emulation, and the real-world testbed node. Java is highly independent from platforms or languages. GENI (Global Environment for Network Innovations) [15] and FIRE (Future Internet Research and Experimentation) [20] are targeting to embed any network experiments, including clean-slate designs. The major difference of [15, 20] from previous network testbeds has the virtualization layer. This will make it possible to embed multiple network experiments in a testbed at the same time. [15, 20] also provide a number of libraries to support easy experiments. These libraries are to lower the barrier of constructing software from scratch.

2.2 Open Source WMN Testbed at KAIST

Numerous wireless mesh network testbeds have mushroomed up in last couple of years. WiMesh (Wireless Mesh) testbed at KAIST [8] is one of those activities. We had deployed campus scale testbed with 56 mesh routers in our office building and in 6 undergraduate dormitory buildings where are more than 1000 residents over 1 km^2 area (Figure 1). The testbed was implemented with open source codes for all communication protocol layers to give convenience to researchers who want to port their ideas on it.

Even though we had designed WiMesh testbed [8] with open architecture, which has open source codes and provides easy distribution/management tool WiVi [16], it is still not easy to transfer from their simulation codes to the testbed implementation due to the difficulty of system programming and inconvenience of our testbed environment. KAIST WiMesh testbed had developed based on Linux system. On the one hand, Linux system is a very useful platform to develop protocols because it has sufficient open source codes and helpful WiKi based development communities. But Linux is inflexible architecture for clean-slate protocol researches. It has traditional layered architecture thus it might be an obstacle for clean-slate protocol implementations. Furthermore, current testbed architecture does not provide multiple experiments at the same time. Thus some noble architecture updating is strongly required on our testbed.

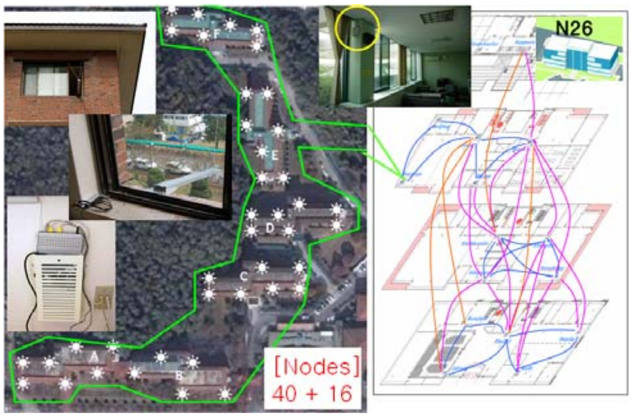

Figure 1. WiMesh (Wireless Mesh) network testbed at KAIST

3. Common Code Architecture

3.1 Common Code Base

The network testbed is a very important element for the network researchers to evaluate their ideas, needless to say, it is mandatory for future Internet researchers who try clean-slate approaches. Our focuses of designing wireless mesh network testbed are:

1) To give a flexibility for implementation of any ideas
2) To give the easiest way for implementation of their codes
3) Virtualization

The easiest way to implement clean-slate ideas on the real system such as a testbed is sharing common code base with network simulator. The common code base is a set of program or algorithm that can be run different software platform or hard architecture. In the sense of development network protocol, a common code base can be run both a software network simulator and a hardware testbed node. Now we define here *common code architecture* as an architecture that utilizes a common code base. If a mesh network testbed node can reuse a code set implemented in a network simulator without any modification. We can say that it is made *common code architecture* based. Common set of metrics, tools for monitoring, and analysis for experiment results are also concerns about *common code architecture* building.

3.2 Redesign of Network Simulator

Network researchers are very familiar with network simulators and mostly they use them to justify their ideas. Then they want to extend their trials into a real world testbed or an emulator to get/show conviction their ideas. With making a close investigation of network simulators and real system network protocol stacks, we found out very a common software set. We exploited it as a common code base. Then we replace the protocol stack of mesh node with the practiced simulator protocol stack.

We choose GloMoSim [13] network simulator to exploit a common code base for a mesh node protocol stack (see Figure 2). Because GloMoSim was intrinsically designed for multi-hop wireless network simulator and it has a variety of dedicated multi-hop wireless protocol stacks. And it is implemented using custom C-language and Parsec [14]. Parsec is just C-based simulation language for parallel computing. Only Parsec codes are used in the GloMoSim kernel. Most GloMoSim users do not need to know how the kernel works. GloMoSim users can develop their codes with writing pure C codes. We can build *common code architecture* for testbed mesh node using the GloMoSim network protocol stack instead of using Linux TCP/IP protocol stacks. It gives user friendly programming environment and realizes easily clean-slate ideas into implementation on real system.

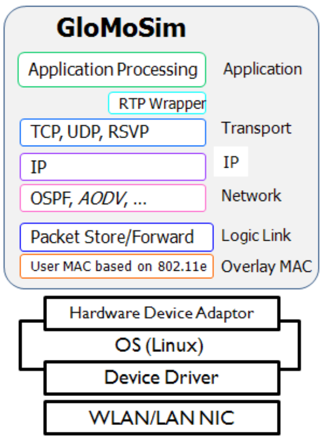

Figure 2. *Common code architecture* that has GloMoSim network protocol stacks.

But there are several problems in order to use a simulator protocol stack as a real network protocol stack. Here we list some modifications of an original GloMoSim simulator and additional work as followings. The first is that redundant parts of simulator must be removed. The simulator also includes radio, mobility, and propagation models. These are redundant codes so that simulator protocol stack is working as a real protocol stack. These unnecessary components must be removed in order to use a simulator protocol stack for a mesh node protocol stack. The second, we must replace simulation clock with real system clock. In all of cases simulators use a simulated clock, which advances in constant increments of time. Constant increment of time is decided by time stamp of the earliest event. As mentioned before, GloMoSim uses Parsec library to build simulation kernel for parallel computing. Parallel simulation requires synchronization of clock among multiple processors. Parsec library is not open source in public domain. Unfortunately we cannot utilize all of original GloMoSim libraries. Therefore it is necessary to build some libraries such as *simclock*, random number generators etc. The third is providing hardware device driver adaptor. In the Linux system (mesh node operating system), only hardware access is available through the device driver. We have to make a special adaptor between the GloMoSim protocol stack and the hardware WLAN NIC device driver. We will discuss details in section 3.3.

3.3 Hardware Device Adaptor

There are a lot of constraints on implementation of MAC layer over the testbed. MAC layer has inherently hardware features and especially 802.11 MACs are mostly proprietary hardware. That means user only can modify MAC functionalities through open source device driver. It is very limited and very hard work to do something on the hardware. Furthermore, most chipset vendors never reveal their hardware features.

We meet a very serious problem for the clean-slate MAC layer researches on the testbed. The first, changing MAC layer for the research requires the use of proprietary hardware that has been never opened. Second even simple experiments such as transmission power control scheme, adjustment of content windows and time synchronized operation must require complicate kernel and device driver programming. These hardships prevent MAC layer researchers from implementing their ideas on a real hardware or testbed.

To achieve our goal, we divide MAC layer into two parts. Those are a hardware MAC and a software MAC. The hardware MAC is an untouchable part of MAC layer built in hardware and firmware. Generally it is a proprietary solution of vendors. The software MAC means a part of MAC layer which we can implement our idea on it just by software programming. It consists of device driver and kernel library such as *mac802.11* of Linux. We want to move it to the user space MAC layer of the GloMoSim protocol stack. Researchers who want to implement clean-slate protocol with MAC modification can port easily their idea on the real system just by user space overlay MAC software programming. They do not need hard program such as device driver and kernel system program.

To make adaptation between a user space overlay MAC and a system program. We designed *hardware device adaptor* layer. The GloMoSim protocol stack can send and receive data packet to/from the WLAN NIC using this adaptor. There are two kinds of *hardware device adaptor* according to its performance. One is *generalized hardware device adaptor* as shown Figure 3. It is designed for easy adaptation to any type of hardware WLAN devices using socket program. It is independent from the type of WLAN devices. Another is *specialized hardware device adaptor* that is targeting performance optimization using *proc file* system and device driver programming.

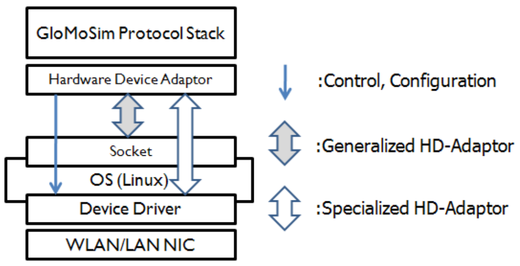

Figure 3. Two types of *hardware device adaptor*

A real application is also applicable to the GloMoSim protocol in the same way. To generate interface between a protocol stack and applications, *application adaptor* can be considered for setting up the virtual tunnels with *tun/tap* device.

3.4 Virtualization

Virtualization is making a single resource to appear as multiple logical resources. All users think that they occupy whole network testbed under the virtualization. We want to evaluate more many clean-slate attempts on our testbed and do not want our testbed engaged in single heavy researcher all the time. Virtualization can provide solution on an ardent desire. Thus virtualization is not an optional choice for the future Internet testbed.

Node virtualization is realized by task switching that guarantee allocated resources. Tasks running in the same machine must be mutually independent. Our node virtualization approach is user level virtualization unlike previous work that manages task switching at the kernel level. Virtual node is a nothing but simulator node that is combination of each layer protocol stacks. We can generate nodes that have different protocol stacks. All of virtual nodes share a common event scheduler. We realize node virtualization using it. GloMoSim schedules event using message queue. We can guarantee the allocated resource of each virtual node using this message queue. A message is delivered to the message queue with weighted time delay. Higher weight means more delayed delivery than others.

4. Conclusion

Clean-slate design approach is a new trend of future Internet research. And a testbed for the future Internet research takes very important position. Virtualization and user friendly programming environment are not optional features of future Internet testbed.

In this paper, we proposed innovative architecture of a wireless mesh testbed node called *common code architecture*. It is adequate to the future Internet testbed that adopt clean-slate attempts. Proposed mesh testbed node architecture minimizes the gap between initial implementation on the simulation environment and extended implementation to the real network testbed. Therefore proposed architecture gives the most user friendly programming environment.

References

[1] A. Feldmann, “Internet Clean-Slate Design: What and Why?”, ACM SIGCOMM Computer Communication Review, Vol.27, No.3, 2007, pp.59-64
[2] M. Kropff, T. Krop, M. Hollick, P. S. Mogre, and R. Steinmetz, “A Survey on Real World and Emulation Testbeds for Mobile Ad Hoc Networks,” TRIDENTCOM 2006.
[3] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracua, H. Liu, and M. Singh, “Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols,” Proc. of WCNC 2005.
[4] G. Judd and P. Steenkiste, “Repeatable and Realistic Wireless Experimentation Through Physical Emulation,” ACM SIGCOMM Computer Communications Review, 34(1):63-68, January 2004.
[5] D. Johnson, T. Stack, R. Fish, D. Flickinger, R. Ricci, and J. Lepreau, “TrueMobile: A Mobile Robotic Wireless and Sensor Network Testbed,” Technical Report FTN-2005-02, University of Utah Flux Group, April 2005.
[6] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, “A Blueprint for Introducing Disruptive Technology into the Internet,” Proc. of HotNet-I, October 2002.
[7] T. Krop, M. Bredel, M. Hollick, and R. Steinmetz, “JiST/MobNet: Combined Simulation, Emulation, and Real-World Testbed for Ad hoc Networks,” WINTECH 2007.
[8] Sachin L. S., J.S. Lee, A.S. Lee, K.H. Lee, J.H. Lee, S. Chong, “An Open Wireless Mesh Testbed Architecture with Data Collection and Software Distribution Platfrom”, TridentCom 2007
[9] Puljiz Z., Mikuc M., “IMUNES Based Distributed Network Emulator”, Software in Telecommunications and Computer Networks, 2006
[10] University of Utah Flux Research Group, “Emulab: The Utah Network Testbed”, http://www.emulab.net/
[11] Eddie Kohler, “The Click Modular Router”, Ph.D Thesis, MIT, Nov. 2000.
[12] http://www.isi.edu/nsnam/ns/
[13] http://pcl.cs.ucla.edu/projects/glomosim/
[14] http://pcl.cs.ucla.edu/projects/parsec/
[15] http://www.geni.net
[16] http://www.scalable-networks.com/
[17] http://www.opnet.com/
[18] http://143.248.238.6/~sachin/realization/
[19] http://pdos.csail.mit.edu/roofnet/docu.php
[20] http://cordis.europa.eu/fp7/ict/fire/