The NCTUns 1.0 Network Simulator

S.Y. Wang, C.H. Huang, C.L. Chou, C.C. Hwang, T.H. Yan, Z.M. Yang, C.C. Chiou, and C.C. Lin

shieyuan@csie.nctu.edu.tw
Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan

Abstract

This paper presents the NCTUns 1.0 network simulator, which is a high-fidelity and extensible network simulator capable of simulating both wired and wireless IP networks. After two years' developments, the quality of the NCTUns 1.0 has become mature and now it can be released to the network community. We have set up a web site at http://NSL.csie.nctu.edu.tw/nctuns.html for people to download this software package. In this paper, we will present the architecture, design, and implementation of the NCTUns 1.0 network simulator as well as many of its useful features.

1. Development History

The NCTUns 1.0 network simulator is a high-fidelity and extensible network simulator capable of simulating various protocols used in both wired and wireless IP networks. Its core technology is based on the novel simulation methodology invented by S.Y. Wang at Harvard University in 1999 [1, 2]. Due to this novel methodology, the NCTUns 1.0 network simulator provides many unique advantages that cannot be achieved by traditional network simulators such as OPNET [3] and ns-2 [4].

The predecessor of the NCTUns 1.0 network simulator is the Harvard network simulator [5], which was authored by S.Y. Wang in 1999. Since its release in July 1999, as of January 1, 2002, the Harvard network simulator has been downloaded by more than 2,000 universities, research institutes, industrial research laboratories, and ISPs.

As user feedbacks about the usages of the Harvard network simulator gradually come back, it becomes clear that the Harvard network simulator has several limitations and drawbacks that need to be overcome and solved, and some useful features and functions need to be added and implemented. For these reasons, after joining National Chiao Tung University (NCTU), Taiwan in February 2000, S.Y. Wang designed a new simulation methodology for the NCTUns 1.0 network simulator. Since that time, S.Y. Wang has been leading his students in his Network and System Laboratory (NSL) to design and implement the NCTUns 1.0 network simulator until now.

The NCTUns 1.0 network simulator is based on the new simulation methodology. As such, many limitations and drawbacks with the Harvard network simulator are removed. It uses a distributed architecture to support remote simulations and an open-system architecture to enable protocol modules to be easily added to the simulator. In addition, it has a fully-integrated GUI environment for editing a network topology and specifying network traffic, plotting performance curves, configuring the protocol stack used inside a network node, and playing back animations of logged packet transfers. In all aspects, the NCTUns 1.0 network simulator is much more powerful and useful than the Harvard network simulator.

After two years' developments, the quality of the NCTUns 1.0 network simulator has become mature and thus it can be released to the network community. To promote its uses, we have set up a web site at http://NSL.csie.nctu.edu.tw/nctuns.html for people to download this software package, fetch documentations, join mailing lists, and participate in discussions. More information about these details can be found at the web site listed above. Figure 1 shows the starting screen of the NCTUns 1.0 network simulator.

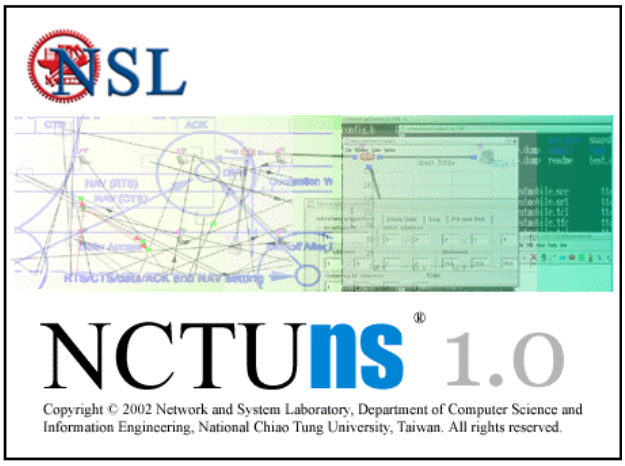

Figure 1: The starting screen of the NCTUns 1.0 network simulator.

2. Motivation

Network simulators implemented in software are valuable tools for researchers to develop, test, and diagnose network protocols. Simulation is economical because it can carry out experiments without the actual hardware. It is flexible because it can, for example, simulate a link with any bandwidth and propagation delay or a router with any queue size and queue management policy. Simulation results are easier to analyze than experimental results because important information at critical points can be easily logged to help researchers diagnose network protocols.

Network simulators, however, have their limitations. Because a complete network simulator not only needs to simulate hosts and routers, but also needs to simulate application programs that generate network traffic and network utility programs that configure, monitor, or gather statistics about a simulated network, developing a complete network simulator is a large effort. Due to limited development resources, traditional network simulators usually have the following drawbacks:

- Simulation results are not as convincing as those produced by real hardware and software equipment. In order to constrain their complexity and development cost, most existing network simulators can only simulate real-life network protocol implementations with limited detail, and this may lead to wrong results. For example, OPNET's modeler product [3] uses a simple finite state machine model to model complex TCP protocol processing. As such, the accuracy of the resulting simulation results may be questioned.
- These simulators are not extensible in the sense that they lack the standard UNIX POSIX application programming interface (API). As such, existing or to-be-developed real-life application programs cannot run normally to generate traffic for a simulated network. Instead, they must be rewritten to use the internal API provided by the simulator (if there is any) and be compiled with the simulator to form a single big and complex program. For example, since the ns-2 [4] network simulator itself is a user-level program, there is no way to let another user-level application program "run" on top of it. As such, a real-life application program cannot run normally to generate traffic for a network simulated by the ns-2 program.

To overcome these problems, we proposed a simulation methodology in [1, 2] and used it to implement the Harvard network simulator. The Harvard network simulator has two desirable properties as follows. First, it uses the real-life UNIX TCP/IP protocol stack, real-life network application programs, and real-life network utility programs. As such, it can generate more accurate simulation results than a traditional TCP/IP network simulator that abstracts a lot away from a real-life TCP/IP implementation. Second, it lets the system default UNIX POSIX API (i.e., the standard UNIX system call interface) be provided on every node in a simulated network. Any real-life UNIX application program, either existing or to be developed, thus can run normally on any node in a simulated network to generate traffic. One important advantage of this property is that since an application program that is developed for simulation study is a real UNIX program, the program's simulation implementation can be its real implementation on an UNIX machine. As a result, when the simulation study is finished, we can quickly implement the real system by reusing its simulation implementation.

Although the methodology proposed in [1, 2] can provide the above two advantages, it has several limitations and drawbacks. To remove these problems, we designed a new simulation methodology and used the new methodology to design and implement the NCTUns 1.0 network simulator. In the rest of the paper, we will present the new simulation methodology as well as the features, components, architecture, design, and implementation of the NCTUns 1.0 network simulator.

3. Main Components of the NCTUns 1.0 Network Simulator

The NCTUns 1.0 network simulator uses a distributed architecture to support remote simulations and concurrent simulations on multiple simulation machines. Functionally, it can be divided into eight separate components described below:

- The first component is the fully-integrated GUI environment by which a user can edit a network topology, configure the protocol modules used inside a network node, specify mobile nodes' moving paths, plot performance curves, play back animations of logged packet transfers, etc. The GUI program uses Internet TCP/IP sockets to communicate with other components. As such, the GUI program can run on a machine that is far away from the simulation machine.

  From the network topology edited by the user, the GUI program can generate a simulation job description file suite and submit it to the remote simulation machine for execution. When the simulation is finished, the simulation results and generated log files are transferred back to the GUI program. The GUI program then can either examine logged data, plot performance curves, or play back packet transfer animations, etc. While a simulation is running at the remote simulation machine, the GUI user can query or set an object's value at any time. For

example, the GUI user may query/set the routing table of a router or the switch table of a switch at any time. If the GUI user does not want to do any query or set operation during a simulation, the GUI user can choose to disconnect the currently running simulation so that he/she can use the GUI program to handle other simulations. The GUI user can later reconnect the disconnected simulation at any time, no matter whether the simulation is finished or not. A GUI user thus can submit many simulation jobs in a short period of time. Doing this can increase simulation throughput if there are many simulation machines available to service these jobs concurrently.

- The second component is the simulation engine. A simulation engine is a user-level program. It functions like a small operating system. Through a defined API, it provides useful and basic simulation services to protocol modules. Such services include virtual clock maintenance, timer management, event scheduling, variable registrations, etc. The simulation engine needs to be compiled with various protocol modules to form a single user-level program, which we call the "simulation server." The simulation server takes the simulation job description file suite as its input, runs the simulation, and generates logged data and packet transfers as its output. A simulation server is a process. It is created to service a job. When the job is finished, it terminates.
- The third component is various protocol modules. A protocol module is like a layer of a protocol stack. It performs a specific protocol or function. For example, the ARP protocol or a FIFO queue is implemented as a protocol module. A protocol module is composed of a set of functions. It needs to be compiled with the simulation engine to form a simulation server. Inside the simulation server, multiple protocol modules can be linked into a chain to form a protocol stack.
- The fourth component is the simulation job dispatcher, which is a process and always alive. It is used to manage and use multiple simulation machines at the same time to increase simulation throughput. The job dispatcher can sit between a large number of GUI users and a large number of simulation servers. When a GUI user submits a simulation job to the job dispatcher, the dispatcher will select an available simulation machine to service this job. If there is no available machine at this time, the submitted job can be queued in the dispatcher as a background job. Background jobs are managed by the dispatcher and can be serviced using various scheduling policies.
- The fifth component is the coordinator. On every machine where a simulation server program resides, a process called the "coordinator" exists. The coordinator process is alive as long as the simulation machine is alive. When a simulation machine is powered on and brought up, the coordinator running on that machine will register itself with the dispatcher to join the dispatcher's simulation machine farm. Later on, when its status (idle or busy) changes, it will notify the dispatcher of its new status. This enables the dispatcher to choose an available machine from its simulation machine farm to service a job.

  When the coordinator receives a job from the dispatcher, it forks a simulation server to simulate the specified network and protocols. It may also fork several real-life application programs specified in the job, which are used to generate traffic for the simulated network. When the simulation server is alive, the coordinator communicates with the dispatcher and the GUI program on behalf of the simulation server. For example, periodically the simulation server sends the value of the current virtual clock of the simulated network to the coordinator. The coordinator then forwards this information to the GUI program. This enables the GUI user to know the progress of the simulation. During a simulation, the GUI user can also on-line set or get an object's value (e.g., to query or set a switch's current switch table). Message exchanges that happen between the simulation server and the GUI program are all done via the coordinator.
- The sixth component is the kernel source patches that need to be made to the kernel of the UNIX machine so that a simulation server can correctly run on it. In our simulation methodology, some kernel source files need to be modified. For example, during a simulation, TCP timers need to be triggered by the virtual clock of the simulated network rather than by the real clock.
- The seventh component is various protocol daemons running at the user level. Like the routing daemon "routed" or "gated" running on UNIX machines that exchange routing messages and set up system routing tables, when the NCTUns 1.0 network simulator is running to simulate a network, various protocol daemons can run at the user level for various purposes. For example, the real-life "routed" (using the RIP routing protocol) or "gated" (using the OSPF routing protocol) daemons can run with the NCTUns 1.0 network simulator to set up the routing tables used by the nodes in a simulated network.
- The last component is all real-life application programs running at the user level. As stated previously, any real-life user-level application program can run on a simulated network to either generate network traffic, configure network, or monitor network, etc. For example, the tcpdump program can run on a simulated network to capture packets flowing over a link and the traceroute program can run on a simulated network to find out the routing path traversed by a packet flow.

Figure 2 depicts the distributed architecture of the NCTUns 1.0 network simulator. It shows that, due to the nature of the distributed architecture, simulation machines can be very far away from the machines where the GUI programs run. For example, the simulation service center may be at NCTU in Taiwan while the GUI users come from many places of the world.

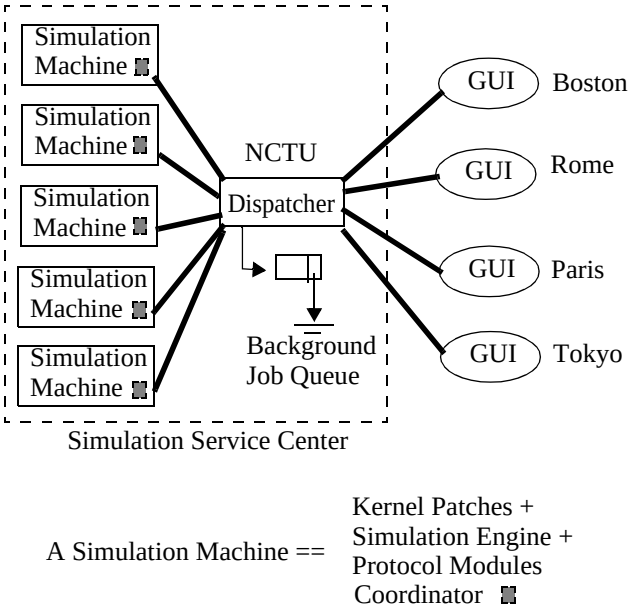

Figure 2: The distributed architecture of the NCTUns 1.0 network simulator.

When the components of the NCTUns 1.0 network simulator are run on multiple machines to carry out simulation jobs, we say that the NCTUns 1.0 network simulator is operating in the "multiple machine" mode. This mode can support the "remote simulation" and "concurrent simulation" functions. These components can also run on the same machine to carry out simulation jobs. This mode is called the "single-machine" mode and is more suitable for a user who has only one machine. Due to the nature of the Inter-Process Communication (IPC) design, the NCTUns 1.0 network simulator can be used for either mode without changing its program code. Only the mode parameter in its configuration file needs to be changed.

4. Main Components of the Fully-Integrated GUI Environment

The NCTUns 1.0 network simulator has a fully-integrated GUI environment by which a user can easily perform simulation studies. The GUI program is composed of four main components. In the following, we will present each of them.

The first component is the topology editor, which is shown in Figure 3. The topology editor provides a convenient and intuitive way to graphically construct a network topology, specify various parameters of network devices and protocols, and specify the application programs that will be run during simulation to generate traffic. A constructed network can be either a fixed wired network or a mobile wireless network. Due to a user-friendly design, all GUI operations can be done easily and intuitively.

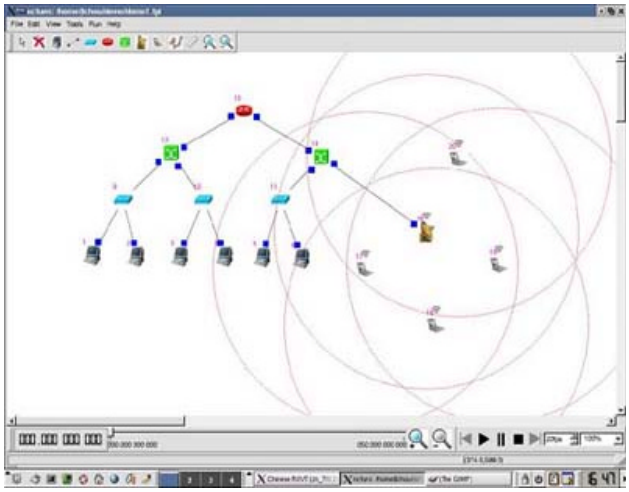

Figure 3: The topology editor of the NCTUns 1.0 network simulator.

The second component is the performance monitor, which is shown in Figure 4. The performance monitor can easily and graphically display the plots of some monitored performance metrics such as a link's utilization or a TCP connection's achieved throughput.

The third component is the node editor, which is shown in Figure 5. A node in the NCTUns 1.0 network simulator represents a network device such as a switch or an IEEE 802.11 wireless LAN access point. The node editor provides a convenient environment to flexibly configure the protocol modules used inside a network node. By using this tool, a user can use the mouse to graphically add, delete, or replace a protocol module with his/her own module. As such, the node editor enables a user to easily test the functionality and performance of a new designed protocol. Figure 5 shows the internal protocol stacks used by a wireless LAN access point, which has three network interface ports. In Figure 5, each square box represents a protocol module. We can see that each network interface port is configured with a chain of protocol modules (i.e., configured with a protocol stack). The protocol modules supported by the NCTUns 1.0 network simulator are classified into different categories (e.g., MAC, PHY, Packet Scheduling, etc.) and are displayed on the left-hand side of the node editor.

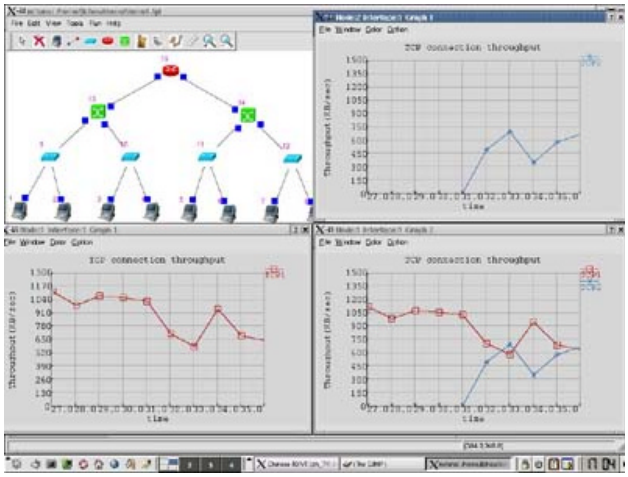

Figure 4: The performance monitor of the NCTUns 1.0 network simulator.

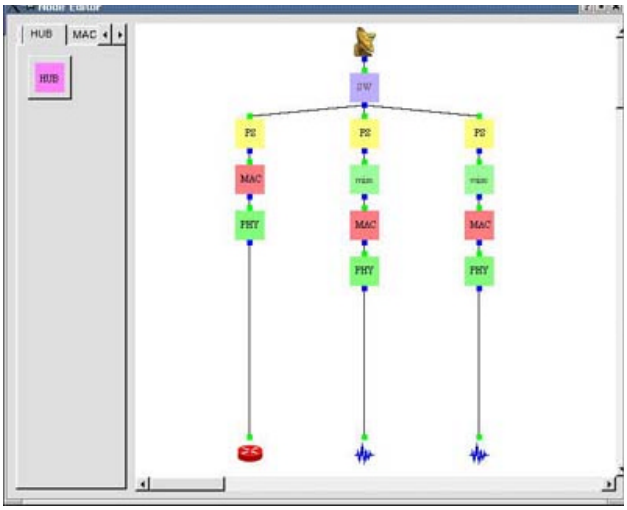

Figure 5: The node editor of the NCTUns 1.0 network simulator.

The last component is the packet animation player, which is shown in Figure 6. By using the packet animation player, a logged packet transfer trace can be graphically replayed at any speed. Both wired and wireless networks are supported. The network at the top of Figure 6 is a fixed wired network. When the packet animation player starts, packets are represented as line segments with arrows and flow smoothly on the links. The network at the bottom is a mobile ad hoc network. When the packet animation player starts, a wireless transmission is represented by two circles centered at the transmitting node. These two circles represent the transmission and interference ranges of the wireless network interface. Their display duration is proportional to the packet transmission time of this wireless transfer. The packet animation player is a very useful tool because it can help a researcher to visually debug and test the behaviors of a protocol. It is also very useful for educational purposes.

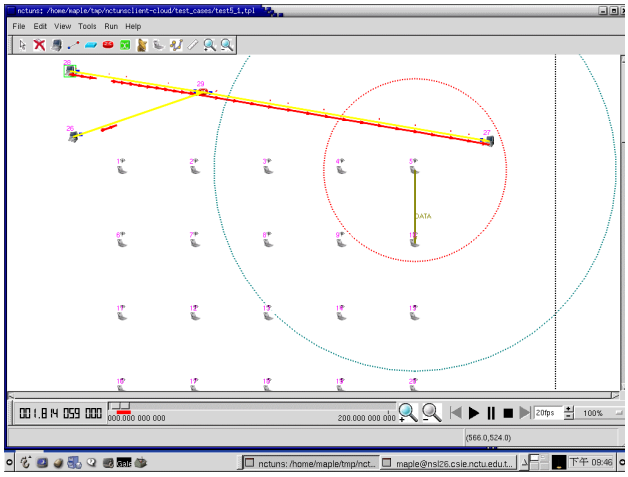

Figure 6: The packet animation player of the NCTUns 1.0 network simulator.

5. Network Simulation Capability

Due to the open-system architecture of the NCTUns 1.0 network simulator, new or so-far-unsupported network protocols and functions can be added to the simulator easily. They can be implemented as either protocol modules or protocol daemons. By using the API provided by the simulation engine, a protocol module developer does not need to know the implementation details of the NCTUns 1.0 network simulator. As such, the developer can focus more of his/her time on the protocol designs.

We classify the network simulation capabilities of the NCTUns 1.0 network simulator into two categories. The first is “network devices.” The second is “network protocols.” Table 1 lists the types of network devices that are currently supported. Table 2 lists the network protocols that are currently supported, which are sorted based on layers.

Device	Acronym	Technology Specification
Host	HOST	For wired networks
Hub	HUB	For wired networks
Switch	SWITCH	For wired networks
Router	ROUTER	For wired networks
IEEE 802.11 Mobile Node	MNODE	For mobile wireless networks
IEEE 802.11 Access Point	AP	For mobile wireless networks

Table 1: The types of network devices that are currently supported by the NCTUns 1.0 network simulator.

Physical Layer

Protocol	Acronym	Technology Specification
Full-Duplex Point-to-point Link	FDP2P	For wired networks
Half-Duplex Point-to-point Link	HDP2P	For wired networks
Half-Duplex Shared Medium	Air	For wireless networks

Data Link Layer

Protocol	Acronym	Technology Specification
Ethernet	Ether	IEEE 802.3
Wireless LAN (ad hoc mode)	WLANAD	IEEE 802.11b
Wireless LAN (infra-structure mode)	WLANIN	IEEE 802.11b
Learning Bridge Protocol (used in the switch)	LBP	IEEE 802.11d
Spanning Tree Protocol (used in the switch)	STP	IEEE 802.11d
Address Resolution Protocol	ARP	RFC-826

Table 2: The network protocols that are currently supported by the NCTUns 1.0 network simulator. (sorted based on layers)

Network Layer (continued from Table 2)

Protocol	Acronym	Technology Specification
Internet Protocol (unicast and broadcast)	IP	RFC-791, RFC-792, RFC-826
Internet Control Message Protocol	ICMP	RFC-792
Open Shortest Path First Routing protocol	OSPF	RFC-1247
Routing Information Protocol	RIP	RFC-1058
Fixed Network God Routing Protocol	FNGRP	Automatically calculate the best routing paths in a fixed network (for research purposes)
Dynamic Source Routing protocol	DSR	For mobile ad hoc networks
Ad hoc On Demand Distance Vector Routing protocol	AODV	For mobile ad hoc networks
Adaptive Distance Vector Routing protocol	ADV	For mobile ad hoc networks
Destination-Sequenced Distance-Vector Routing protocol	DSDV	For mobile ad hoc networks
Mobile Network God Routing Protocol	MNGRP	Automatically calculate the best routing paths in a mobile ad hoc network (for research purposes)
FIFO Packet Scheduling mechanism	FIFO	(Can be used in a switch as well)
Round-Robin Packet Scheduling mechanism	RR	(Can be used in a switch as well)

Transmission Layer

Protocol	Acronym	Technology Specification
Transmission Control Protocol	TCP	RFC-791, RFC-792, RFC-826
User Datagram Protocol	UDP	RFC-768, RFC-1122

Application Layer

Protocol	Acronym	Technology Specification
HTTP		(Note: All existing real-life
FTP		network application programs
Telnet		or tools can directly run on a
Tcpdump		simulated network. Here listed
Traceroute		are just a few of them.)
Ping		

6. Simulation Methodology

The NCTUns 1.0 network simulator's simulation methodology is based on the kernel-reentering technique proposed in [1, 2]. Using this technique, both the Harvard and NCTUns 1.0 network simulators can achieve two unique advantages. The first advantage is using the real-life TCP/IP protocol stack to generate simulation results while the second advantage is using real-life application programs to generate network traffic.

Despite this similarity, the simulation methodology used by the NCTUns 1.0 network simulator is much more powerful and useful than that used by the Harvard network simulator. The enhancements come from the desires to simulate various network devices and protocols, simulate various types of networks, support both broadcast and unicast transfer modes for application programs, let users use the familiar real-life IP address and port number scheme to specify the network parameters of application programs, etc. In short, the goal of the new simulation methodology is to enable users to specify any desired simulated network and operate it in exactly the same way as they operate a physical real network. The Harvard network simulator fails to achieve this goal because (1) it can simulate only simple fixed networks consisting of hosts and routers; and (2) it requires users to use a specially-designed IP address and port number scheme to specify the network parameters of application programs, which confuses most users.

To achieve this goal, the new simulation methodology needs to modify several parts of the UNIX kernel source code. For example, to let an application program automatically use the virtual time of the simulated network rather than the real time when performing time-related operations such as setting up an alarm timer, the dispatcher registers the process ID of the forked application program with the kernel by issuing a system call into the kernel. After the registration, all future time-related system calls issued by this registered application program will use the virtual time of the simulated network kept in the kernel rather than the real time. This design avoids the need to get and modify the source code of the application program. In contrast, the Harvard network simulator requires the source code of an application program to be available and modified so that time-related system calls can be redirected to special system calls that use the virtual time rather than the real time.

Due to space limitation, in this introductory paper we have no room to present the new simulation methodology and the kernel modifications that are required to support this new methodology. These design and implementation details will be presented in another paper. To let readers understand the kernel re-entering technique, however, in the following, we will use a simple example network depicted in Figure 7 to briefly illustrate this technique.

6.1. Tunnel Network Interface

Tunnel network interfaces is the key facility in the simulation methodology. A tunnel network interface, available on most UNIX machines, is a pseudo network interface that does not have a real physical network attached to it. The functions of a tunnel network interface, from the kernel's point of view, are no different from those of an Ethernet network interface. A network application program can send out its packets to its destination host through a tunnel network interface or receive packets from a tunnel network interface, just as if these packets were sent to or received from a normal Ethernet interface.

Each tunnel interface has a corresponding device special file in the /dev directory. If an application program opens a tunnel interface's special file and writes a packet into it, the packet will enter the kernel. To the kernel, the packet appears to come from a real network and just be received. From now on, the packet will go through the kernel's TCP/IP protocol stack as an Ethernet packet would do. On the other hand, if the application program reads a packet from a tunnel interface's special file, the first packet in the tunnel interface's output queue in the kernel will be dequeued and copied to the application program. To the kernel, the packet appears to have been transmitted onto a network and this pseudo transmission is no different from an Ethernet packet transmission.

Using tunnel network interfaces, we can easily simulate the single-hop TCP/IP network depicted in Figure 7 (a), where a TCP sender application program running on host 1 is sending its TCP packets to a TCP receiver application program running on host 2. We set up the virtual simulated network by performing the following two steps. First, we configure the kernel routing table of the simulation machine so that tunnel network interface 1 is chosen as the outgoing interface for the TCP packets sent from host 1 to host 2, and tunnel network interface 2 is chosen for the TCP packets sent from host 2 to host 1. Second, for the two links to be simulated, we run a simulation server to simulate them. For the link from host i to host j (i = 1 or 2, j = 3 - i), the simulation server opens tunnel network interface i's and j's special file in /dev and then executes an endless loop until the simulated time elapses. In each step of this loop, it simulates a packet's transmission on the link from host i to host j by reading a packet from the special file of tunnel interface i, waiting the link's propagation delay time plus the packet's transmission time on the link, and then writing this packet to the special file of tunnel interface j.

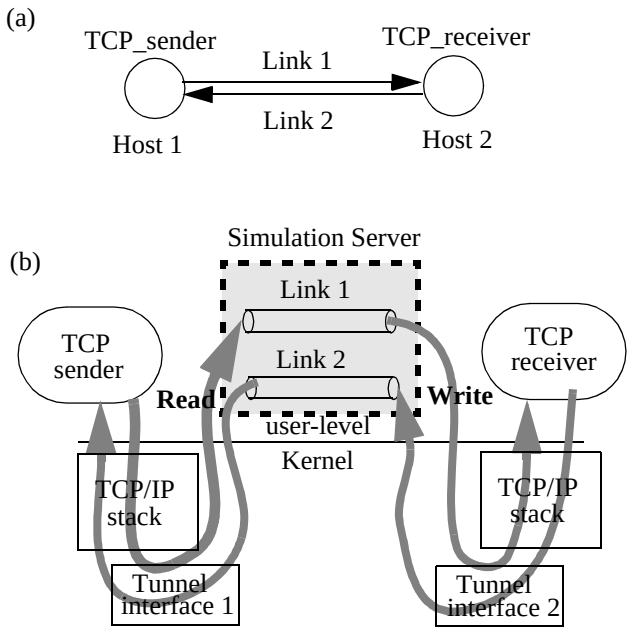

Figure 7: (a) A TCP/IP network to be simulated. (b) By using tunnel interfaces, only the two links need to be simulated. The complicated TCP/IP protocol stack need not be simulated. Instead, the real-life TCP/IP protocol stack is directly used in the simulation.

After performing the above two steps, the virtual simulated network has been constructed. Figure 7 (b) depicts this simulation scheme. Since the trick of replacing a real link with a simulated link happens outside the kernel, the kernels on both hosts do not know that their packets actually are exchanged on a virtual simulated network. The TCP sender and receiver programs, which run on top of the kernels, of course do not know the fact either. As a result, all existing real-life network application programs can run on the simulated network, all existing real-life network utility programs can work on the simulated network, and the TCP/IP network protocol stack used in the simulation is the real-life working implementation, not just an abstract or a ported version of it.

Note that in this simulation methodology, the kernel of the simulation machine is shared by all nodes (hosts and routers) in a virtual simulated network. Therefore, although in Figure 7 (b) there are two TCP/IP protocol stacks depicted, actually they are the same one -- the protocol stack of the single simulation machine.

7. Capability Comparison between the Harvard and NCTUns 1.0 Network Simulators

Based on a new simulation methodology, the NCTUns 1.0 network simulator removes many limitations and drawbacks with the Harvard network simulator. The NCTUns 1.0 network simulator uses a distributed and open-system architecture. This architecture design greatly enhances its extensibility and system functionality. With the use of a fully-integrated GUI environment, the NCTUns 1.0 network simulator not only enables users to more easily perform simulation studies, but it also has a chance to hide its many internal details and non-real-life usages from the users. Table 3 compares the capabilities of the Harvard network simulator with those of the NCTUns 1.0 network simulator. From the table, we see that the NCTUns 1.0 network simulator is much more powerful and useful than the Harvard network simulator in all aspects.

Network Simulation Capability	Harvard Network Simulator	NCTUns 1.0 Network Simulator
Can application programs use unicast and broadcast modes to transfer packets?	No. Application programs can use only the unicast mode to transfer packets.	Yes. Beside the unicast mode, an application program can broadcast a packet to multiple application programs.
Can the simulator support having multiple subnets in a simulated network?	No. The whole simulated network must be a subnet.	Yes.
Can the simulator simulate hosts, hubs, switches, routers, mobile nodes, access points, and other various network devices?	No. The simulator can simulate only hosts and routers.	Yes.
Can the simulator simulate shared medium like air or half-duplex mode links?	No. The simulator can simulate only full-duplex point-to-point links.	Yes.
Can routing daemons be run to automatically generate routing tables for a simulated network?	No, not provided.	Yes. Right now, RIP and OSPF routing daemons can be run to automatically generate routing tables for a simulated network.
Can the simulator simulate mobile ad hoc networks?	No.	Yes. Node mobility is supported.

Table 3: Capability comparison between the Harvard and the NCTUns 1.0 network simulator.

(continued from Table 3)

Can the modules used inside a node's protocol stack be easily modified and replaced?	No. The modules used inside a node's protocol stack cannot be modified or replaced.	Yes. By using the GUI node editor, a node's protocol stack can be graphically and easily modified.
Does an application program need to be modified before it can correctly generate traffic in a simulated network?	Yes. An application program's source code needs to be modified and recompiled.	No. An application program's source code need not be modified. Its execution code can be run readily to correctly generate traffic in a simulated network.
Can the simulator support concurrent simulations on multiple machines?	No. The simulator can run only one simulation at one time.	Yes, because it uses a distributed architecture.
Does the simulator have an integrated GUI environment?	No.	Yes.
Can the simulator support remote simulations?	No.	Yes, because it uses a distributed architecture.
Can the simulator log wired and wireless packet transfers and later use a packet animation player to replay them?	No.	Yes.
Can new protocol modules be easily added to the simulation engine to enhance its simulation capability?	No.	Yes, because it uses an open-system architecture.
Do the IP addresses reported in an application program's output use the real-life IP address scheme and meaning?	No. The IP addresses reported in an application program's output use a special IP address scheme that has a different meaning than the real-life	Yes.
Does the simulator have a performance monitor to easily plot performance curves?	No.	Yes.
Can application programs bind to the same port number when they run on different nodes in a simulated network?	No. Application programs must bind to different port numbers even when they are run on different nodes in a simulated network.	Yes.
Can application programs use the real-life IP address scheme and meaning to send packets?	No. Application programs must use a special IP address scheme and meaning to send packets to a destination node.	Yes. Through the use of the fully-integrated GUI environment, non-real-life usages are hidden from the users.

8. Related Work

In the literature, some approaches also use a real-life TCP/IP protocol stack to generate results [6, 7]. However, unlike our approach, these approaches are used for emulation purposes, rather than for simulation purposes. Among these approaches, Dummynet [7] most resembles our simulator. Both Dummynet and our simulator use tunnel interfaces to use the real-life TCP/IP protocol stack on the simulation machine. However, there are some fundamental differences. Dummynet uses the real time, rather than the simulated network's virtual time. Thus the simulated link bandwidth is a function of the simulation speed and the total load on the simulation machine. As the number of simulated links increases, the highest link bandwidth that can be simulated decreases. Moreover, in Dummynet, routing tables are associated with incoming links rather than with nodes. As such, the simulator does not know how to route packets generated by a router, as they do not come from any link

OPNET [3] and ns [4] represent the traditional network simulation approach. In this approach, the thread-supporting event scheduler, application programs that generate network traffic, utility programs that configure, monitor, or gather statistics about a simulated network, the TCP/IP protocol implementation on hosts, the IP protocol implementation on routers, and links are all compiled together to form a single user-level program. Due to the enormous complexity, such a simulator tends to be difficult to develop and verify. A simulator constructed using this approach cannot provide UNIX POSIX API for real-life application programs to run normally to generate network traffic. Although some simulators may provide their own internal API, real-life application programs still need to be rewritten so that they can use the internal API, be compiled with the simulator successfully, and be concurrently executed with the simulator during simulation.

ENTRAPID [6] uses another approach. It uses the virtual machine concept [8] to provide multiple virtual kernels on a physical machine. Each virtual kernel is a process and simulates a node in a simulated network. The system calls issued by an application program are redirected to a virtual kernel. As such, UNIX POSIX API can be provided by ENTRAPID and real-life application programs can be run in separate address space normally. However, because the complex kernel needs to be ported to and implemented at the user level, many involved subsystems (e.g., the file, disk I/O, process scheduling, inter-process communication, virtual memory subsystems) need to be modified extensively. As such, the porting effort is very large and the correctness of the ported system may need to be extensively verified.

9. Concluding Remarks

After two years' developments, the quality of the NCTUns 1.0 network simulator has become mature and it

now can be released to the network community. The NCTUns 1.0 network simulator is much more useful and powerful than its predecessor -- the Harvard network simulator, which was released in 1999. To promote its uses, we have set up a web site at http://NSL.csie.nctu.edu.tw/nctuns.html for people to download this software package. To let people quickly feel how the NCTUns 1.0 network simulator operates without first setting up a simulation machine on their own, we also have set up a simulation service center at NSL for people to submit and run their simulation jobs on NSL's simulation machines. More information about these details can be found at the above web site.

References

[1] S.Y. Wang and H.T. Kung, "A Simple Methodology for Constructing Extensible and High-Fidelity TCP/IP Network Simulators," IEEE INFOCOM'99, March 21-25, 1999, New York, USA.

[2] S.Y. Wang and H.T. Kung, "A New Methodology for Easily Constructing Extensible and High-Fidelity TCP/IP Network Simulators," accepted and to appear in "Computer Networks" Journal.

[3] OPNET Inc. (http://www.opnet.com).

[4] S. McCanne, S. Floyd, ns-LBNL Network Simulator. (http://www-nrg.ee.lbl.gov/ns/).

[5] Harvard TCP/IP network simulator 1.0, available at http://www.eecs.harvard.edu/networking/simulator.html.

[6] X.W. Huang, R. Sharma, and S. Keshav, "The ENTRAPID Protocol Development Environment," IEEE INFOCOM'99, March 21-25, 1999, New York, USA.

[7] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," Computer Communication Review, Vol. 27, No. 1, p.31-41, January 1997.

[8] A. Meyer and L.H. Seawright, "A Virtual Machine Time-Sharing System," IBM Systems Journal, Vol. 9, No. 3, 1970, pp. 199-218.