SMI: SNT / NTO / Sun Labs Internship Progress Report: Mobile IPv6 Summer 2003

Author: Ioan Raicu

Document Created:
5/30/2003

Last Date Modified:
9/1/2004

Table of Contents

This document has been edited for proprietary information that is protected by a non-disclosure agreement between Sun Microsystems Inc. and Ioan Raicu.

1.0 Introduction

Mobile IPv6 is a protocol which allows nodes to remain reachable while moving around in the IPv6 Internet. Each mobile node is always identified by its home address, regardless of its current point of attachment to the Internet. While situated away from its home, a mobile node is also associated with a care-of address, which provides information about the mobile node's current location. IPv6 packets addressed to a mobile node's home address transparently cache the binding of a mobile node's home address with its care-of address, and to then send any packets destined for the mobile node directly to it at this care-of address. To support this operation, Mobile IPv6 defines a new IPv6 protocol and a new destination option. All IPv6 nodes, whether mobile or stationary can communicate with mobile nodes. [2]

In order to complete some performance tests on Mobile IPv6 protocol, we will need two different things to be in place and complete. The first item is the ability to look at network traffic and analyze Mobile IPv6 traffic as it is generated. This is very important since Mobile IPv6 is still an experimental work in progress, and therefore it still has many problems; having the ability to look at the Mobile IPv6 traffic will give us a good tool to diagnose what exactly is happening in the network. At the moment, Snoop supports IPv6, but does not support all the extension headers of Mobile IPv6; we will be adding the Mobile IPv6 support to Snoop, a real time packet sniffer under Solaris 10. The second item will be to construct a test-bed where we can use the new Mobile IPv6 protocol. We will set up a testbed where we are going to install 3rd party Mobile IPv6 implementations to work together. This includes setting up various computers in the testbed as home agents, corresponding nodes, and mobile nodes. We will examine two different implementations of mobile IPv6 for our testbed, one being the Kame project [4], while the other is MIPL [5]. The Kame project requires one of the following OS (FreeBSD 3.5, BSD/OS 4.1, NetBSD 1.4.2, OpenBSD 2.7), while the MIPL project requires a Linux kernel 2.4.x or newer (ex. Linux Redhat 7.2 or newer). We will use these two implementations to build our testbed. We will use the testbed to generate the necessary traffic to verify that the Snoop functionality is what we expected it to be.

Once all this is complete, we would like to take some performance measurements, in order to make some good educated guesses about the benefits of Mobile IPv6, and perhaps any drawbacks that this new protocol might bring. The benefits of the route optimization is one key experiment we want to complete, and the scalability limitations of the corresponding node implementation since it requires a rather complex set of message exchange in order to perform binding updates. These results will be very valuable to the ongoing effort to implement the various parts of Mobile IPv6, and make necessary corrections to the protocol design itself early before it is deployed throughout the Internet.

All source code, documentation that is discussed in this report, and final presentation can be found at “/net/snt.eng/export/d56/ws/iraicu”.

2.0 Snoop Modifications - Mobile IPv6

In order to verify that our testbed configuration worked properly, and bring Snoop up to date to support Mobile IPv6, we came to the conclusion we should add the necessary support for Mobile IPv6 into Snoop. I used the Mobile IPv6 Extension Socket API [1] and the IETF Mobile IPv6 Draft [2] to guide me through the Snoop modifications.

2.1 Details of Implementation and Source Code

The following files have been modified/created to add the Mobile IPv6 support in Snoop.

- in.h (2 lines added, 0 lines deleted)
- ip6.h (11 lines added, 0 lines deleted)
- ip6mh.h (187 lines added, 0 lines deleted)
- icmp6.h (71 lines added, 0 lines deleted)
- Makefile (3 lines added, 3 lines deleted)
- readme.txt
- snoop_ip.c (55 lines added, 3 lines deleted)
- snoop_icmp.c (125 lines added, 5 lines deleted)
- snoop_ip6mh.c (481 lines added, 0 lines deleted)
- Makefile (1 line added, 1 lines deleted)

Below is a more detailed description of the changes that were made in each file.

- in.h
  - Defined a few constants
- snoop_ip.c
  - Routing Header Type 2
  - Home Address Destination Option Header
- ip6.h
  - Routing Header Type 2 Definition
- readme.mip6.txt
  - Mobile IPv6 Implementation Detail text file
- ip6mh.h
  - Implemented the API from [1]
  - Defined many constants
  - Defined the following headers
    - ip6_mh: IPv6 Mobility Header
    - ip6mh_home_test_init: Home Address Test Init Message
    - ip6mh_careof_test_init: Care of Test Init Message
    - ip6mh_home_test: Home Address Test Message
    - ip6mh_careof_test: Care of Test Message
    - ip6mh_binding_request: Binding Refresh Request Message
    - ip6mh_binding_update: Binding Update Message
    - ip6mh_binding_ack: Binding Acknowledgement Message
    - ip6mh_binding_error: Binding Error Message
    - ip6mh_opt_refresh_advice: Binding Refresh Advice
    - ip6mh_opt_altcoa: Alternate Car-of Address
    - ip6mh_opt_nonce_index: Nonce Indices

        - ip6mh_opt_auth: Binding Authorization Data
- icmp6.h
    - Defined many constants
    - mip_dhreq: Dynamic HA Address Discovery Request
    - mip_dhrep: HA Address Discovery Reply
    - mip_prefix_solicit: Mobile Prefix Solicitation
    - mip_prefix_advert: Mobile Prefix Advertisements
    - nd_opt_advinterval: Advertisement Interval Option
    - nd_opt_homeagent_info: Home Agent Information
- Makefile
    - Added “ip6mh.h”
- snoop_icmp.c
    - Implemented Advertisement Interval Option and Dynamic HA Address Discovery Request
- snoop_ip6mh.c
    - Implemented most of the Mobile IPv6 functionality
        - Interpret_ip6mh(): IPv6 Mobility Header
        - Prt_ip6mh_back(): Binding Acknowledgement Message
        - prt_ip6mh_berror(): Binding Error Message
        - prt_ip6mh_bu(): Binding Update Message
        - prt_ip6mh_brr(): Binding Refresh Request Message
        - prt_ip6mh_cot(): Care of Test Message
        - prt_ip6mh_coti(): Care of Test Init Message
        - prt_ip6mh_hot(): Home Address Test Message
        - prt_ip6mh_hoti(): Home Address Test Init Message
- Makefile
    - Added “snoop_ip6mh.o”

There are still a few things that need to be complete, but I expect to have most of these things completed by the time I complete my internship.

- Code Review
    - Functionality needs to be tested first
- Test functionality
    - both the test-bed and the traffic generator are not making good progress, and the chances of testing the Snoop functionality completely is beginning to look slim
- Finish a few last details on ICMP6
    - Specifically supporting an arbitrary number of IPv6 HA addresses in the HA Address Discovery Reply.
- WebRev
    - Functionality needs to be tested first
- Sample Output
    - Functionality needs to be tested first
- Demo Configuration
    - Test-bed needs to be finished first

3.0 Mobile IPv6 Installation Instructions

The next few sections describe the series of steps to install the OS on the machines that will be part of the final testbed. They are in no way comprehensive, and might require some external or a-priory knowledge regarding Unix like OS installation, and network configuration. These instructions should only be used as a basic guide.

I wrote up installation instructions on how to install Linux, enable all the services (ftp, telnet, etc…), install IPv6, and enable the router functionality for both IPv4 and IPv6. The installation files have been saved under "mip-linux.os-install.txt"; the mobile IPv6 stack (MIPL) installation instructions will be found at "mip-linux.mip6-configure.txt".

3.1 Testbed overview

The MIPL (Mobile IPv6 for Linux) IPv6 implementation was tested in Linux Redhat 7.2 and Debian / Woody systems with various 2.4.x kernel versions. The KAME IPv6 implementation was tested in variations of BSD OS; one of the supported platforms is FreeBSD 4.8. Note that due to the early and experimental implementations of mobile IPv6, it is a good idea to install the same OS and version that the developers used for testing, and upgrading to the latest version of a particular OS might cause many more problems in an installation in an already error prone and non-trivial installation process.

The testbed configuration is depicted below in Figure 1. Some of the abbreviations throughout this document and the figure below are: MN – mobile node, CN – corresponding node, and HA – home agent.

We managed to get three x86 machines that we can use to load Linux Redhat 7.2 OS and the MIPL mobile IPv6 implementation. Two of the machines had unusable hard drives since the machines were too old (1997) and the hard drives were too new (1999), and the bios would just not support such large drives. We eventually got some external SCSI drives that were older and that worked in these older systems. We installed Linux on all three machines and finished up some basic configurations. After having some problems configuring the MN machine, we got another x86 machine that was newer, and installed Linux Redhat 9.0 OS. In order to perform the necessary experiments, we needed to have at least two different subnets, so we also got a SPARC machine on which we loaded Solaris 10, and we configured as a router.

Therefore, our testbed consists of 4 machines, including a laptop that has not been configured yet:

- Name: mip6-linux7
  - Functionality: CN
  - Architecture: x86
  - OS: Linux Redhat 7.2, Kernel 2.4.20

- Name: mip6-linux8
  - Functionality: HA
  - Architecture: x86
  - OS: Linux Redhat 7.2, Kernel 2.4.20
- Name: mip6-linux10
  - Functionality: MN
  - Architecture: x86
  - OS: BSD
- Name: mip6-linux11
  - Functionality: MN
  - Architecture: x86
  - OS: Linux Redhat 9.0, Kernel 2.4.20
- Name: mip-ha1
  - Functionality: CN
  - Architecture: x86
  - OS: Linux Redhat 7.2, Kernel 2.4.20
  - IPv6 Gateway: it uses 6to4 tunneling to access the outside world

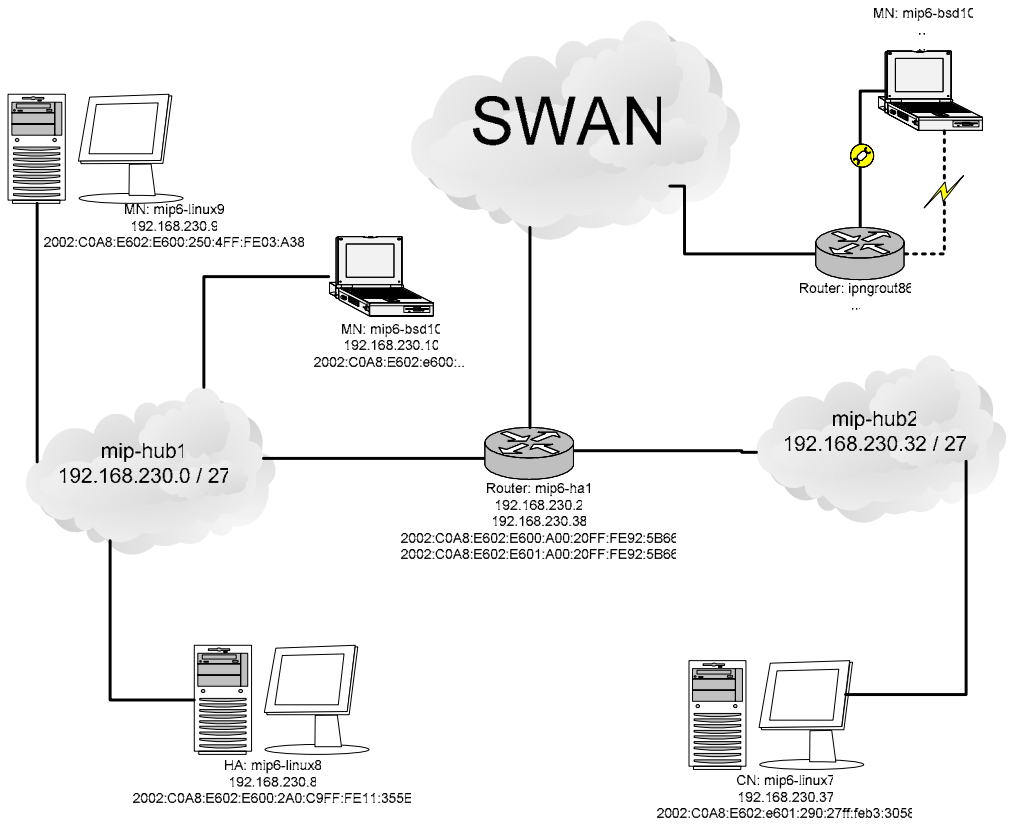

Figure 1: Mobile IPv6 Testbed – Logical Overview

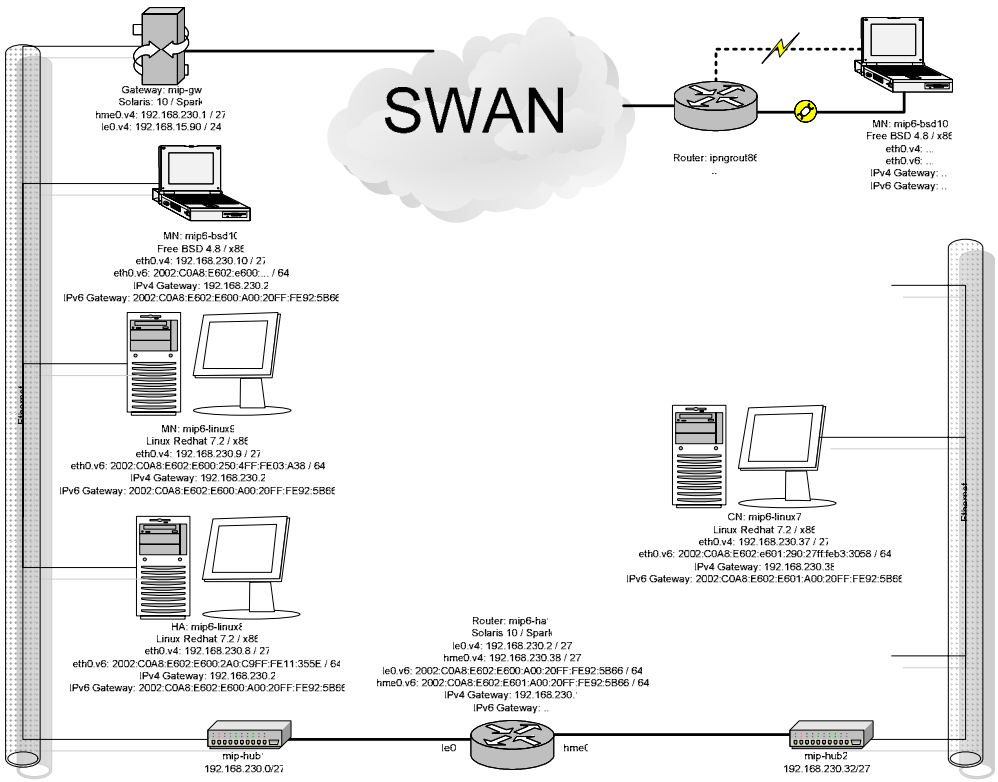

Figure 2: Mobile IPv6 Testbed – Physical Overview

The testbed configuration is still not functioning properly. The current state is that all the various Mobile IPv6 components (MN, HA, CN, and Router) are installed properly and seem to work. When the MN attempts to move from one network (the home network) to another (the foreign network), it should send the binding update (BU) to the HA; however, the BU is never sent because the MN cannot find the tunnel that goes back to the HA. IPv6-in-IPv6 tunneling is enabled, and the tunnel exists when ifconfig is used. The end points to the tunnel should be configured from the "/etc/sysconfig/network-mip6.conf" file via the HOMEADDRESS and HOMEAGENT field.

We also ran into some problems due the use of 6to4 addresses (2002:<IPv4 address / 32>:<subnet ID / 16>:<host id / 64>) in the testbed. Since SWAN is set up to use 6to4 addresses, we decided to set up our testbed using 6to4 addresses and therefore the entire testbed would be globally reachable. This addressing scheme gave us problems on the Linux machines with respect to the default route functionality. The default route would not work properly, and the only way to get packets to be forwarded to the correct network would be to add entries into the routing table with a prefix length of 48 bits, or longer. By assigning site local addresses (fec0:: / 16), the default route problems we had fixed themselves; we did not try other global reachable addresses (2000:: / 3) to see if they pose the same problems 6to4 addresses posed.

Some things that should be tried in order to resolve some of these issues and get the testbed working properly are:

- Reassign all network addresses using global addresses (2000::/3) instead of site local or 6to4 addresses; this solves the default route problem we experienced with 6to4 addresses

- Reconfigure router to advertise as the default route the global / site local address instead of the link local address; the link local address is identical between all networks on the same router, and therefore either the global or the site local addresses must be advertised on each respective link in order for the movement detection algorithm to work correctly in Mobile IPv6
    - The MN must be able to differentiate when it moves from one network to another; the old default route should not be accessible anymore, and be different from the new default route

3.2 OS Installation

3.2.1 Linux Redhat 7.2

See "mip-linux.os-install.txt". For configuration purposes, the Linux IPv6 HOWTO written by Peter Bieringer is an excellent document [6].

3.2.2 FreeBSD 4.8

FreeBSD 4.8 can be downloaded from

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/mirrors.html

3.3 Mobile IPv6 Testbed

3.3.1 MIPL (Mobile IPv6 for Linux)

Detail instructions can be found in "mip-linux.mip6-configure.txt". The "Mobile IPv6 Linux-HOWTO" written by Lars Strand is also another excellent source of information when it comes to configuring the Mobile IPv6 testbed [8]. Yet another great tutorial is "The Linux Kernel HOWTO" [9] written by Alavoor Vasudevan; this is needed for patching the kernel with mobile IPv6 support, configuring the kernel, recompiling it, installing the kernel and modules, and any other details revolving around the Linux kernel configuration.

3.3.2 KAME

More information about the Kame Mobile IPv6 stack, please refer to http://www.kame.net/ [4]. The KAME Mobile IPv6 stack works with Free BSD.

4.0 Mobile IPv6 Performance Evaluation Overview

Since we were not able to finish the testbed configuration and get the Mobile IPv6 protocol stack to function properly, we were not able to do any performance measurements. In case we would have finished the testbed in time, we would have liked to perform the following tests.

Initially, we would have taken some base line measurements from the testbed. The crucial part for our performance analysis is the ability to create an application (simulator) that can generate mobile IPv6 traffic, such as binding updates (BU), etc… This simulator is a necessary part of our work due to the inability to generate scenarios where there are many mobile clients to put any kind of real world burden on the corresponding node (CN) and home agents (HA). We will use this simulator to load both the CN and HA with BU while we are performing similar measurements as the baseline ones of measuring the various metrics for the mobile node (MN) and how the performance relates to the CN and HA. This simulator in reality will be a traffic generator that can generate the appropriate Mobile IPv6 traffic to suit the needs of our performance measurements, and will be mainly developed by Hanlong Wang (hanlong.wang@Sun.COM), another intern also working with Samita Chakrabarti (Samita.Chakrabarti@Sun.COM).

4.1 Simulation Overview

The baseline measurements would be to compare the response times between the MN and CN while it is on the home network and while it is on the foreign network. While it is home, the MN to CN packets should be traveling the most direct route possible. After the MN node moves to another network, the MN packets will be tunneled through the HA and onto the CN. This difference will be interesting to have as the baseline measurements.

These tests require a few programs:

- P1: A simulation program/daemon which can determine that MN has moved; it would be useful for printing correct report information as well.
- P2/P3: A client-server software suite that sends data (TCP) and receives it in a sink as well. It should also measures per-packet response time while the MN is on the home network, and when the MN moves to the new network.
- P4: A traffic generator

The client/server would be operating on a specific port number and P1 would be able to detect the interface configuration and deduce that the MN has moved; it would then send a trigger to the client which would then start the time measurements for a given period.

A packet generator (TG) is required which can send BU to the HA. It should also be able to generate packets for return routability and route optimization. Thus the packet generator will have the ability to send COTI/HOTI messages, process COT/HOT messages, and then sends BU to the CN. The TG should use a raw streams device to open the Ethernet driver to write the Ethernet+IPv6+MH data directly to the Ethernet driver. The TG should also be able to process all these messages as well. This program would be very useful and could be used to test basic CN functionality. More details about this TG are given in section 4.2.

A few example test scenarios are described below:

- Populate CN with n BCE entries.
    - Use a program which generates IRE caches for random IPv6 addresses (contact Ashish Mehta for such test program).
    - Use the packet generator to simulate BU for those n home addresses.
    - Move MN from home network to CN network / a different network and measure packet response time
- Run the simulator program to send BU for n home addresses while moving the MN and start measuring the response time
- The packet generator sends m Bus, delete a subset of them by sending BU with life-time 0, and then again send a few BUs to create new ones; move MN during this period and take measurements

4.2 Mobile IPv6 Support in NS2

Due to time constraints, Mobile IPv6 was not tested for performance, as we intended to do at the beginning of the summer. Another alternative to actual testing would be to attempt to simulate Mobile IPv6 within the defacto-standard simulator NS2 that is used for mostly any network performance simulations. Until recently, this would have been very difficult since there was no support for Mobile IPv6 within NS2, which would have meant that whoever tried to do any simulation runs would have to implement the Mobile IPv6 stack within NS2.

Well, recently Thierry Ernst [10] has announced the release of the NS-2.1b6 enhancements to simulate Mobile IPv6 in large Wide-Area networks within the NS2 simulator. This code has been developed by MOTOROLA Labs Paris and INRIA Rhone-Alpes PLANETE team. The code can be downloaded from http://www.inrialpes.fr/planete/mobiwan/. Presently, the tar file comprise of:

- Mobile IPv6 and simple IPv6 extensions needed by Mobile IPv6
- A library to create, configure, and manipulate large topologies in a very easy manner (TOPOMAN)
- A new translator from GT-ITM to NS-2 which output TOPOMAN procedure calls
- Embedded extensions to allow the simultaneous use of wireless features, mobility, and multicast

Mobile IPv6 is available, in two modes:

- Local mobility (i.e. within an administrative domain, i.e. within a bounded area)
- Global mobility (crossing administrative domains, i.e. moving from one side of the topology to another).

Hierarchical Mobile IPv6 will later be included in the distribution and will be used to demonstrate global mobility. This offers a very good environment to test the limits of Mobile IPv6, especially the scalability of the CN and HA under scenarios where the number of MN are astronomical (100K, millions, etc…). It is true because of the complexity of the simulation environment, the number of nodes within the network might be limited to far fewer nodes (less than 100K), however even those results can be somewhat extrapolated to come up with some meaningful results for much larger networks.

4.3 Traffic Generator (Simulator)

The Traffic Generator (TG) is being designed with extensibility in mind to have the ability to add any new protocols, and control all field values in any of the protocols being used. In reality, for this project, only the necessary protocols in order to support Mobile IPv6 will be implemented, but the extensibility will be desirable in order for the TG to be usable in other projects in the future. An overview of current existing traffic generators, the protocols they support, and platforms can be found in my report for the Neon project [11].

Mobile IPv6 has not been implemented in Solaris yes, and therefore there is no API that we can use to write an application that will use the Mobile IPv6 headers. If we were to use RAW IP sockets, which would allow us to concentrate solely on the Mobile IPv6 headers and extension headers, we would have no way of specifying to the IPv6 layer that we now have a new protocol (0x62). Because raw IP sockets sits on top of IPv6, we would have no way to modify any fields in the IPv6 layer. We therefore were forced to implement the traffic generator from scratch and implement each of the protocols headers and field values using DLPI. This sounds like a lot of work, but it really is better than what it sounds like. We use the header definitions of all these protocols (that have already been defined), and have a function for each protocol that goes through each field and initializes the corresponding fields with an appropriate values, and then we have a packet assembly function that takes all the various headers (structures) and copies them in a buffer in the order that they will appear when they are send over the wire. The raw device (ex. eth0) is open for writing, and the buffer (ex. Ethernet, IP, TCP, payload) is send over the raw device. This model is both extensible (new protocols can be added relatively easy), and is fully configurable (all fields in all layers are accessible). We would like to make a configuration file from which the TG could take all of its traffic specification, including the protocols, and the default/randomized field values.

Figure 3 below shows the high level overview of the TG and the various components. Note that each of the protocols is initialized independently including the payload, and at the end they are each copied in some particular order into a buffer that is then send out using DLPI over the raw device. The supported protocols/extension headers that are being implemented are:

- Ethernet
- IPv6
- IPv6 Mobility Header
  - Home Address Test Init Message (HOTi)
  - Care of Test Init Message (COTi)
  - Home Address Test Message (HOT)
  - Care of Test Message (COT)
  - Binding Refresh Request Message (BRR)
  - Binding Update Message (BU)
  - Binding Acknowledgement Message (BACK)
  - Binding Error Message (BERROR)

If time permits, it would be interesting to implement some of the ICMP6 functionality; the following protocols/extension headers could be implemented.

- ICMP6
  - Dynamic HA Address Discovery Request
  - HA Address Discovery Reply

- Mobile Prefix Solicitation
- Mobile Prefix Advertisements
- Advertisement Interval Option
- Home Agent Information

- Routing Header Type 2
- Home Address Destination Option Header

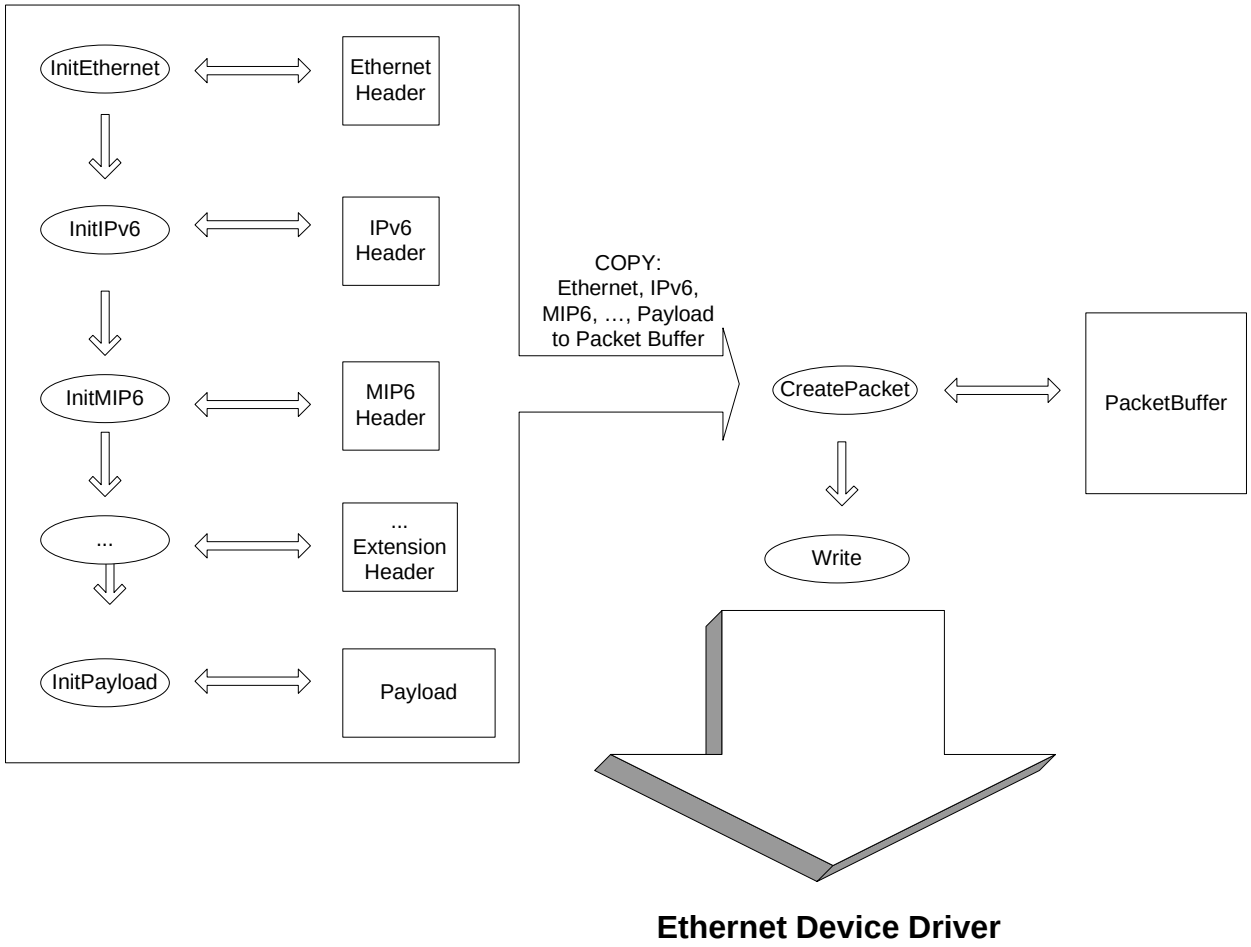

Figure 3: Traffic Generator Overview; the ovals represent function calls, while the rectangles represent memory buffer space, and the arrows represent flow of functionality and data.

The state of the traffic generator is still very experimental, and most likely will not be ready by the time I leave. The advantage for having the TG as we described it here in this document is the fact that any field within any protocol is fully customizable, and we could now send packets with specific properties (such as a HOTi/COTi message from Mobile IPv6) which we could not otherwise have sent by itself, since from the application layer, we would have no way of actually generating such a packet that is normally part of the protocol itself, and is usually triggered by a BU.

We hope that this traffic generator with an open architecture can serve as a good basis for other traffic generators and be used in other projects within Sun at the very least.

4.4 Hash Tables vs. Tries

All the components of Mobile IPv6 (MN, CN, and HA) all use has tables to perform their binding updates (BU). The BU is a critical part of mobile IPv6 and the performance of the BU will affect the performance of mobile IPv6 under heavy loads. There needs to be the ability to insert, find, and delete entries into the particular BU tables fast. Both hash tables and tries offer a linear time asymptotic solution, but the constant factor will undoubtedly be different, and the kind of data that will be stored in these tables is very important. Because of the importance of these BU, I wrote a small section to address the performance of hash tables vs. tries. The results presented here are from a study that Sunay Tripathi (Sunay.Tripathi@eng.sun.com) performed.

A general overview of hash tables pros and cons are:

- Insert performed in O(1) on average
- Delete performed in O(1) on average
- Find performed in O(1) on average
- It is relatively simply to implement
- Table size is fixed
- It is expensive to resize the table size because all elements need to be rehashed
- Inefficient operation if table fills close to capacity
- Depends on good hash function
    - Hash function depends on the distribution of the data
    - In a real world system, the distribution of the data cannot always be guaranteed, and therefore the performance of hash tables cannot be guaranteed
- Cannot do prefix matching, or longest prefix matching as being done in most IP routers

A general overview of tries pros and cons are:

- Insert performed in O(1) in worst case
- Delete performed in O(1) in worst case
- Find performed in O(1) in worst case
- It is more complex to implement
- If the implementation is array based, the memory consumption is much more than the actual data
- If the implementation is link list based, then the space complexity is reduced to the same size as the data itself, but the performance depends on the length of the alphabet, and will most likely not be able to stay within the O(1) asymptotic performance
- If the trie is heavily populated, then the link list implementation could consume twice as much memory than data due to the both the pointers and the data for each element
- Can perform both prefix matches and exact matches within the O(1) time complexity
- The trie is deterministic regardless of the input, which means its performance can be accurately predicted regardless of the distribution of the data
- A trie can hold the entire complete data set, so there would be no resizing necessary under any circumstance

Sunay Tripathi performed some simulations comparing a path compressed trie with a hash table. The tests were done on inserts, lookup, and delete, and the data that was being manipulated was a 32 bit IPv4 address and two 8 bit source / destination ports. The hash table calculated its key from all three fields (48 bits), while the trie used 3 tries to index the source address and

source/destination ports. The data set was generated using Park & Miller random number generator (both for source IP address and ports). The simulation was done for connections ranging from 1000 to 1 million. The results are depicted below in Figure 4 and Figure 5.

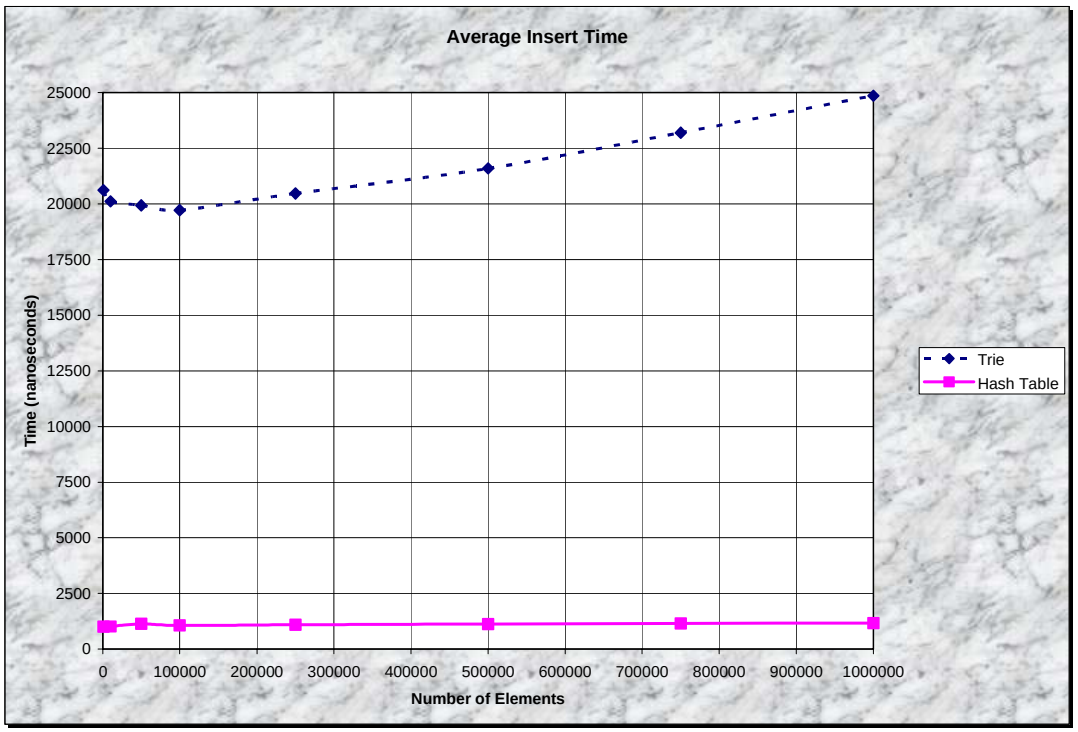

Figure 4: Average Insert Time in nanoseconds between the trie and the hash table

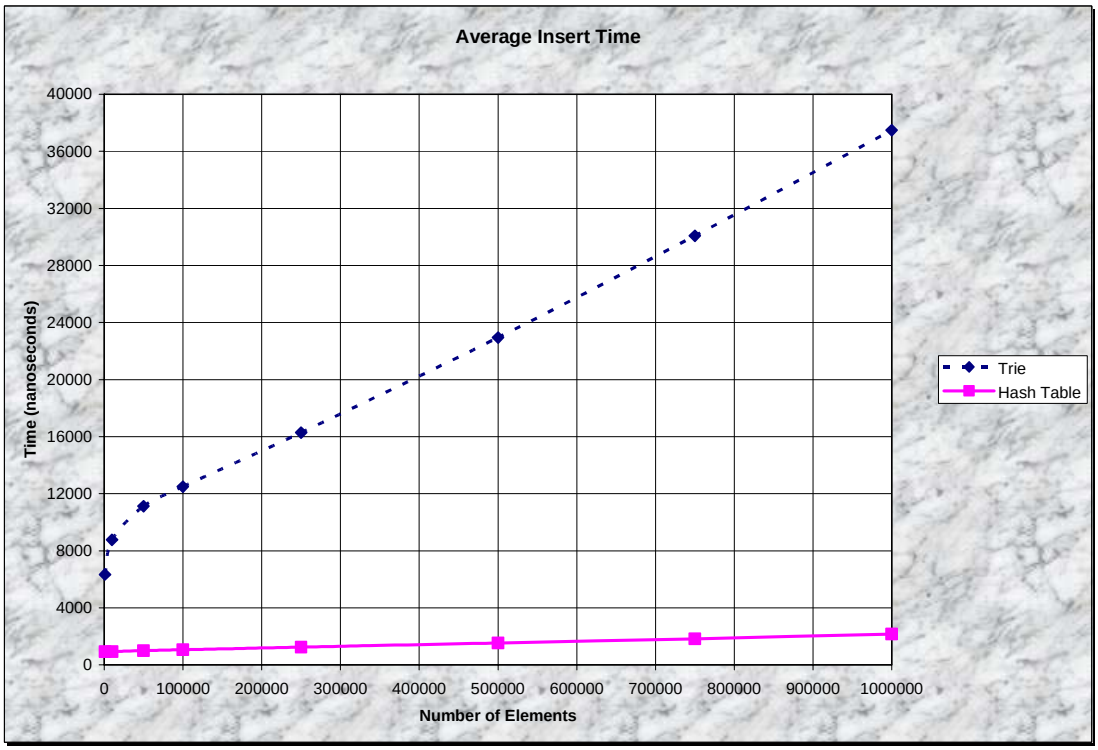

Figure 5: Average Find Time in nanoseconds between the trie and the hash table

5.0 Conclusion

The entire Mobile IPv6 project was a great experience! I learned a whole lot about the difficulties encountered while using 3rd party software that is in very early state of development of protocols that are evolving daily and have not been standardized. I hope that my experiences and this report is enough to get someone else that much closer to completing this work and get some empirical results regarding Mobile IPv6's performance.

The state of the work that I did is the following:

- Testbed
  - Completed
    - Installed the OS and basic configuration for IPv6
    - Configured both the kernel and user level applications for the MN, CN, and HA
  - Future Work
    - Fix the configuration of the testbed so the movement detection algorithm will work properly
- Snoop
  - Completed
    - Added Mobile IPv6 support to Snoop (almost 1000 lines of code written)
  - Future Work
    - Finalize snoop testing
    - Do code review and putback
- Performance Testing
  - Completed
    - Wrote up some tests that should be done
  - Future Work
    - Do baseline measures of Mobile IPv6 using the testbed
    - Do a scalability measure using the testbed and the traffic generator designed
    - Do a scalability study of Mobile IPv6 using NS-2 Simulator
- Traffic Generator (TG)
  - Completed
    - Skeleton of the TG is implemented with support for Ethernet / IPv6 / Mobile IPv6
  - Future Work
    - Finish coding
    - Test against snoop and ethereal
    - Generate scripts for traffic patterns to be useful in simulating environment in the performance evaluation of the Mobile IPv6 testbed

The overview of the Mobile IPv6 protocol status as far as Sun and IETF is concerned is that it is very close from becoming an RFC and therefore a standard. As for Sun, the CN functionality has the highest priority so it can leverage form the fact that Sun's main market is the server market, and therefore it is the key component of Mobile IPv6 that must be implemented.

6.0 Bibliography

[1] S. Chakrabarti, E. Nordmark. "Extension to Sockets API for Mobile IPv6 < draft-chakrabarti-mobileip-mipext-advapi-01.txt>," Internet Engineering Task Force, June 2003

[2] D. Johnson, C. Perkins, J. Arkko. "Mobility Support in IPv6 <draft-ietf-mobileip-ipv6-24.txt>," Internet Engineering Task Force, June 2003

[3] Visual Slick Edit (http://www.slickedit.com/)

[4] KAME Mobile IPv6 Stack. (http://www.kame.net/)

[5] MIPL Mobile IPv6 Stack. (http://www.mipl.mediapoli.com/)

[6] Peter Bieringer. "Linux IPv6 HOWTO", pb@bieringer.de, June 2003.

[7] Sun Microsystems. "IPv6 Administration Guide", April 2003.

[8] Lars Strand. "Mobile IPv6 Linux-HOWTO", lars@unik.no, July 31, 2003.

[9] Alavoor Vasudevan. "The Linux Kernel HOWTO", alavoor@yahoo.com, May 2003.

[10] Thierry Ernst. "NS-2.1b6 enhancements to simulate Mobile IPv6 in large Wide-Area networks", http://www.isi.edu/nsnam/ns/ns-contributed.html, July 2003.

[11] Ioan Raicu. "Neon Internship Report," August 2003.