NSSA - RIT

# FCT Experimental Code Base

**Alan Meekins**

**5/28/2010**

Overview of all utilities and library routines developed to facilitate experimental deployments of the Floating Cloud Tiered Internet Architecture

# Testbed

## OVERVIEW

The local testbed is composed of twelve commodity PCs running Ubuntu linux version 8.04.1 with kernel version 2.6.24. Each computer has one network connection to the control network which is used to allow secure command line access to each machine and to serve as backbone for file transfers between machines for setting up experiments. The remaining interfaces are direct crossover connections between computers forming the topology shown below. Network experiments may use all network interfaces except for the control network to form connections between machines. All IP addresses on the testbed are statically assigned and should not be changed. Node1 is the only testbed machine which is outwardly accessible; all other machines must be accessed through this machine.

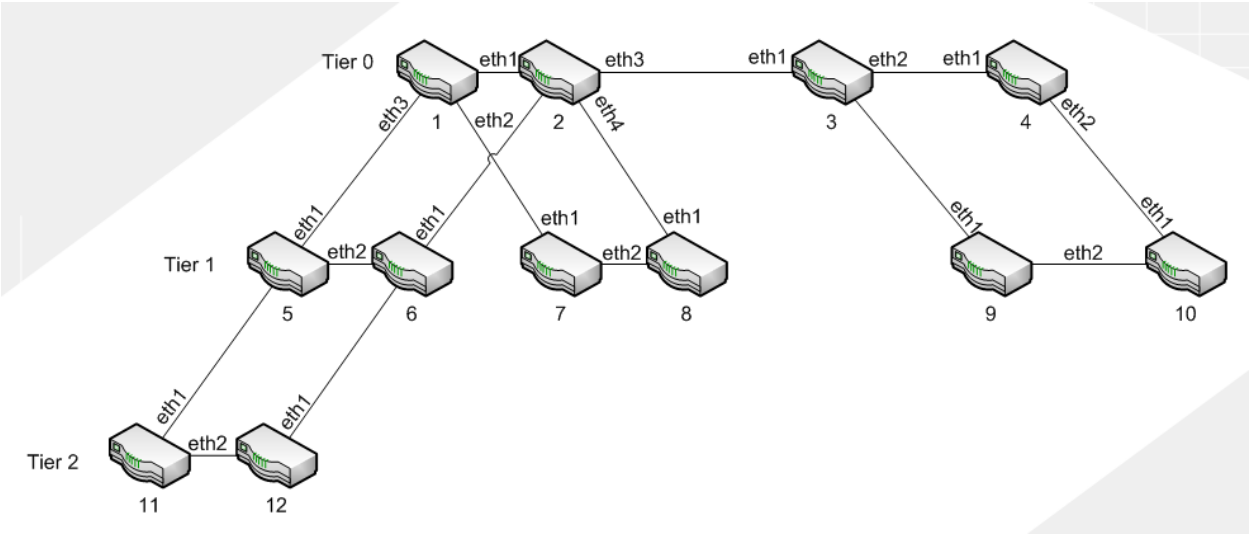


Figure 1: Testbed topology

| Host Name | Eth0(Control) | Eth1 | Eth2 | Eth3 | Eth4 |
|---|---|---|---|---|---|
| **node1** | 192.168.27.245<br>Eth0:1 = 192.168.0.1 | 192.168.2.1 | 192.168.2.5 | 192.168.2.9 | - |
| **node2** | 192.168.0.2 | 192.168.2.2 | 192.168.2.13 | 192.168.2.17 | 192.168.2.21 |
| **node3** | 192.168.0.3 | 192.168.2.18 | 192.168.2.25 | 192.168.2.29 | - |
| **node4** | 192.168.0.4 | 192.168.2.26 | 192.168.2.33 | - | |
| **node5** | 192.168.0.5 | 192.168.2.10 | 192.168.2.37 | 192.168.2.41 | - |
| **node6** | 192.168.0.6 | 192.168.2.14 | 192.168.2.38 | 192.168.2.45 | - |
| **node7** | 192.168.0.7 | 192.168.2.6 | 192.168.2. | - | - |
| **node8** | 192.168.0.8 | 192.168.2.22 | 192.168.2. | - | - |
| **node9** | 192.168.0.9 | 192.168.2. | 192.168.2. | - | - |
| **node10** | 192.168.0.10 | 192.168.2. | 192.168.2. | - | - |
| **node11** | 192.168.0.11 | 192.168.2.42 | 192.168.2. | - | - |
| **node12** | 192.168.0.12 | 192.168.2. | 192.168.2.46 | - | - |

Figure 2: Testbed network interfaces

## TOOLS

There is a small set of bash shell scripts which can be used to easily copy files to all testbed machines or run any command on all machines.

# Account Setup

## LOCAL TESTBED

Login to node1 as root with password sw1tch3d

```
ssh  root@node1
```

Create account

Set a password

Make home folder on all nodes

Copy passwd to testbed nodes

Copy shadow to all nodes

Login as your new account, create ssh keys

## EMULAB

# Building

## DEBUG BUILD

```
make  all  debug
```

## TEST BUILD

```
make  all  test
```

### RELEASE BUILD

```
make all release
```

# Deployment

### LOCAL TESTBED

To deploy a build on the local testbed

```
make cluster_install [build_type]
```

### EMULAB

```
make emulab_install

ssh node1.fct.fct.emulab.net make -C switchbank1/ all [build_type]
```

# Development

### VERSION CONTROL

The code base has been developed using the Bazaar version control system.

#### Checking Out

#### Committing Changes

Committing changes saves a snapshot of the code you have changed locally. During heavy development commits should be preformed frequently as it gives you more choices on versions to revert back to should you find that you do not wish to save your changes. These changes can be reverted or pushed to the main code repository.

```
bzr commit
```

#### Pushing Changes

When changes are pushed you are saving your local snapshots to the main version control server. This should be done at least daily when the code base is being changed. This ensures that your changes are saved to another computer in the event that the machine you develop on dies or is otherwise unusable.

```
bzr push sftp://alan@planet-backup/srv/Shared/siswitch
```

## DATA FORMATS

### FCT Address

| Field | Type | Size | Description |
|---|---|---|---|
| **Tier** | eth_hdr | 4 bits | Tier level |
| **ChunkSize** | uint16_t | 2 bits | Address chunk size recorded as bits/2 minus one |
| **ChunkValue** | uint16_t | 4-16 bits | |
| **ExtraSize** | | 12 bits | Number of extra address bytes |
| **Extra** | uint8_t[] | ExtraSize bytes | Extra address bytes |

### FCT Packet

| Field | Type | Size | Description |
|---|---|---|---|
| **Eth_Header** | eth_hdr | 14bytes | Ethernet frame headers with proto set to 0x8D00 |
| **Pkt_Type** | uint16_t | 2 bytes | Packet type |
| **Data_Len** | uint16_t | 2 bytes | Length of the payload section |
| **Dst_Addr** | FCT_Address | - | |
| **Src_Addr** | FCT_Address | - | |
| **Payload** | uint8_t[] | Data_Len | |

### Command Packets

Command packets have the Pkt_Type field in the FCT header set to 0x0001. The first byte of payload data contains a four bit command type followed by a four bit command code.

#### CMP_EchoReply 0x00

| Field | Type | Size | Description |
|---|---|---|---|
| **Cmp_type** | uint8_t | 1 | Indicates the type of command packet. Is set to 0x00 for EchoReply |
| **SeqNum** | uint16_t | 2 | Sequence number |

#### CMP_DestUnreachable 0x10

| Field | Type | Size | Description |
|---|---|---|---|
| **Cmp_Type** | uint8_t | 1 byte | Set to 0x20 for this packet |
| **SeqNum** | uint16_t | 2 bytes | |
| **Dst_Addr** | FCT Address | - | Destination which could not be reached |

**NOT IMPLEMENTED**. This packet may be sent in response to any packet which could not be forwarded

#### CMP_EchoRequest 0x20

| Field | Type | Size | Description |
|---|---|---|---|
| **Cmp_Type** | uint8_t | 1 byte | Set to 0x20 for this packet |
| **SeqNum** | uint16_t | 2 bytes | Initially set to 1 |

Any host which consumes an echo request packet must respond with a CMP_EchoReply packet addressed to the source of the request. The reply must also contain the same sequence number as the request.

### CMP_CloudAd 0x30

| Field | Type | Size | Description |
|---|---|---|---|
| **Cmp_Type** | uint8_t | 1 byte | Set to 0x30 for this packet |
| **SeqNum** | uint16_t | 2 bytes | |
| **Hostname*** | string | Data_Len-3 | Optional field for debugging purposes |

Any FCT host which receives a cloud advertisement should update all internal routing structures for later routing usage. Routing table entries may be removed if cloud announcements are not received within a regular interval. If the hostname field is included it can be used for debugging purposes to identify the sending FCT node.

### CMP_Trace 0x40

| Field | Type | Size | Description |
|---|---|---|---|
| **Cmp_Type** | uint8_t | 1 byte | Set to 0x30 for this packet |
| **SeqNum** | uint16_t | 2 bytes | Incremented by each forwarding node, initially set to 1 |
| **RealDst_Addr** | FCT_Address | - | Actual destination to route packet to |

**NOT IMPLEMENTED**. This packet is treated like a CMP_EchoRequest which is routed to RealDst_Addr. The destination address in the FCT header must be set to link local. The receiving node must reply to the source address with a CMP_EchoReply where the sequence number is set to the sequence number as found in the CMP_Trace which it processed. The receiver then increments the sequence number of the CMP_Trace packet and forwards it to the next FCT node in the path to the RealDst_Addr.

### CMP_ForcedCloudAd 0x50

| Field | Type | Size | Description |
|---|---|---|---|
| **Cmp_Type** | uint8_t | 1 byte | Set to 0x50 for this packet |
| **SeqNum** | uint16_t | 2 bytes | Set to 0x00 |
| **Hostname*** | char[] | Data_Len-3 | Optional field for debugging purposes |

Announces an FCT cloud that may not be removed if the last announce timer times out.

### CMP_Config

| Field | Type | Size | Description |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

**PARTIALLY IMPLEMENTED**

## IMPORTANT CLASSES

### Mutex

Implements a convient wrapper around a pthread mutex.

### Periodic

Represents a function which should be executed at a certain regular interval

### Pointer

This is a reference counted pointer(AKA: smart pointer) implementation copied from the CommonC++ library. When an instance of pointer goes out of scope it checks to see if it is the last one which references the allocated buffer and if it is the buffer is de-allocated. It is important to note that Pointer uses free() to release the memory it references. Therefore the buffers it points to should be created using malloc().

```
#include “pointer.h”


uint8_t* data=(uint8_t*)malloc(1500);
Pointer p<uint8_t>(data);
exit(1);
```

### Timer

Records the real, user, and system times for the measured period. Additionally the number of voluntary and involuntary context switches is also counted. Pausing and resuming of timers is supported.

```
#include “timer.h”


Timer t();
t.start();
sleep(5);
if(t.isRunning()){
        t.stop();
}
```

```
t.print();
t.reset();
```

## si_address

```
#include "switched.h"


si_address addr=si_address("3.1:2:3");
```

## si_packet

```
#include "switched.h"


si_address srcAddr("1.2");
si_address destAddr("2.2:3");
string data="Hello World";
si_packet packet(srcAddr, destAddr, data.c_str(), data.size());
```

## si_socket

```
#include "switched.h"


string ifname="lo";
si_socket sock(ifname);


if(sock.isOpen()){
        si_packet rxPacket;
        int rxStatus=sock.recv(rxPacket);
        int txStatus=sock.send(rxPacket);
}
sock.close();
```

## raw_address

## raw_packet

si_node

```
#include “si_node.h”


si_node* myNode=new si_node(si_address(“3.1:2:3”));
myNode->addAllInterfaces();
if(myNode->start()){
        while(canrun && myNode->isRunning()){ sleep(5); }
        if(myNode->isRunning()){
                myNode->stop();
        }
}
delete myNode;
```

## DATA STRUCTURES

# Utilities

## ADDRESS TOOL

Used to display the binary format of a given SI/FCT address, the below image is the output when passed one address. The tool reports the number of address chunks and how many bytes is needed to store the address. The third line is the address in hexadecimal followed by the address in binary. The binary dump is grouped by bytes, below which is the bit number.

```
alm2220@ubuntu:switchbank1$ ./addresstool 3.3:4:2
Read address: 3.3:4:2
Has 3 chunks and 5 bytes
0x0d, 0x35, 0x12, 0x00, 0x00,

00001101 00110101 00010010 00000000 00000000
76543210 76543210 76543210 76543210 765432
```

Addresstool can accept two address as seen in the below screenshot. In this case the hex and binary representations of both addresses are displayed along with the minimized address which would be used to address a packet being sent from the first address to a node with the second address.

```
alm2220@ubuntu:switchbank1$ ./addresstool 2.3:4: 3.3:4:2
Read address: 2.3:4:4
Has 3 chunks and 5 bytes
0x09, 0x35, 0x14, 0x00, 0x00,

00001001 00110101 00010100 00000000 00000000
76543210 76543210 76543210 76543210 765432


Read address2: 3.3:4:2
Has 3 chunks and 5 bytes
0x0d, 0x35, 0x12, 0x00, 0x00,

00001101 00110101 00010010 00000000 00000000
76543210 76543210 76543210 76543210 765432


Minimum Address From [2.3(4):4(4):4(4)] To [3.3(4):4(4):2(4)]
2.2(4)
Has 1 chunks and 4 bytes
0x09, 0x21, 0x12, 0x00,

00001001 00100001 00010010 00000000
76543210 76543210 76543210 76
```

## ANNOUNCER

The announcer tool is used to construct and transmit SI/FCT announcement packets. The tool transmits a special type of announcement, CMP_ForcedCloudAd. Any SI/FCT node which receives an announcement of this type will mark it as non-removable so that table entries for that node will not be removed when they timeout. The tool requires at least one interface to operate on, specified with the –I <ifname> option. To transmit on multiple interfaces give the –i option as many times as needed. To automatically add all available interfaces give the –I flag. **Note:** The option to add all interfaces will add all interfaces, including the control interface.

```
alm2220@ubuntu: switchbank1$  sudo ./announcer
Usage: ./announcer [options]

        -i [interface]      Interface to broadcast on
        -I                  Broadcast on all interfaces
        -a [address]        Base source address *REQUIRED*
        -D [delay]          Time to wait between sending packets, in microseconds. Default is 500us
        -n [hostname]       Hostname used in announcements
        -o [offset]         Address offset
        -d [downlinks]      Number of downlinks to announce, default is 2000
        -t [trunklinks]     Number of trunklinks to announce, default is 2000
        -u [uplinks]        Number of uplinks to announce, default is 2000
        -V                  Verbose output
        -v                  Version information
```

## PACKET BUILDER

The packet builder allows you to construct and transmit a packet by specifying the values for important fields on the command line.

```
alm2220@ubuntu:switchbank1$ ./packetBuilder
Warning! This program must be run as root! Don't be surprised if bad things happen.
Usage: ./packetBuilder <options>

        -t <type>               Packet type number
        -s <source>             Source address
        -d <destination>  Destination address
        -i <interface>          Interface name, defaults to lo
        -h                      Display this help message
        -v                      Version information
```

## SIDUMP

Sidump is a utility which prints information about any packets that it receives. The only option that sidump requires is an interface name to listen for packets on.

```
alm2220@ubuntu:switchbank1$ sudo ./sidump lo
int si_socket::open(std::string) - WARNING, Interface is LOOPBACK
Recv packet: [Destination = 0.2; Source = 1.1; Data = 11 bytes] Tue May 25 08:35:25 2010

Recv packet: [Destination = 0.2; Source = 1.1; Data = 11 bytes] Tue May 25 08:35:26 2010

Recv packet: [Destination = 0.2; Source = 1.1; Data = 11 bytes] Tue May 25 08:35:27 2010
```

## SIPERF

Si perf was inspired by iperf and is a loose clone. Similar to iperf, siperf has two modes of operation client and server. Both modes require that you provide the si address of the host as well as at least one network interface. The address can be assigned directly using the –a option followed by the si address or the address can be auto assigned using the –x option. The automatic address assignment flag requires an argument of “child”, “parent”, “trunk”, or “internal” which causes the computer to announce itself as that type of node to the first si node that it receives an announce packet from.

| Address Assignment Mode | Behavior |
|---|---|
| **Child** | Copies the first received announcement, increments the source address tier value and adds an address field to the end of the source address. |
| **Parent** | Copies the first received announcement, decrements the source address tier value and removes the last address field. |
| **Trunk** | Copies the first received announcement, increments the last address field by one. |
| **Internal** | Copies the first received announcement, makes no changes to the source address. |

## Server Mode

In server mode an address must be supplied using either the –a flag or –x along with any number of interfaces to listen on. Siperf then listens for a specially formatted si packet which contains a four byte sequence number at the beginning of the packet's payload. When a packet with a sequence of one is received siperf then starts a new traffic report and sets the expected sequence number to two. As more packets are received from the same host the number of lost packets is counted. When the transmitting client sends a special termination sequence siperf reports the average rate that packets were received and the number of lost and received packets. This report is also printed if the user terminates the program before the last packet is received.

## Client Mode

In client mode siperf can run one of two tests, round trip time(ping) or a through put test. The client requires that one interface be specified using –i and that an address be assigned using –a. To indicate what node we wish to communicate with –c is used followed by an SI address.

### Throughput

When run in client mode throughput testing is the default behavior of siperf. With no additional arguments siperf will transmit packets which are the maximum size allowed by Ethernet frames(1514 bytes) to the host specified by the –c flag at the highest rate allowed by the selected interface. Every two seconds a report is printed displaying the number of packets sent, the size of the packet, the amount of time it spent transmitting, total amount of data sent in megabytes and lastly the transmission rate in megabits per second. Upon exit totals of the same metrics are displayed. By default this test runs for ten seconds

#### Parameters

Several key parameters may be changed through command line flags, packet size, transmission rate, report frequency and test duration. Packet payload size is set using the the –b option followed by the number of bytes. Packet payload size must be at least 4 bytes, if not specified siperf will make the payload large enough to exactly fill the ethernet frame. The test duration can be set using the –d option followed by a time in milliseconds, defaults to 10,000ms. The interval between transmission rate reports can be set using the –t flag followed by time in milliseconds. This options defaults to 2000ms.

The target transmission rate can be set using the –r flag with the number of packets to transmit per second as the argument. This rate control mechanism works to varying levels of accuracy which depends the overall system load, network card, and system architecture. In most cases it undershoots the target rate by half. It is advisable to experiment with the target rate value until the desired rate is reported.

### Round Trip

Round trip measurements can be performed by passing siperf the –p command line flag. Siperf transmits echo request command packets, CMP_EchoRequest, with the sequence number initially set to 1 and the destination and source appropriately filled in. It then waits for an echo reply from the destination host with the same sequence number as the last sent echo request. If a valid echo reply is received the sequence number is then incremented by one and another echo request is transmitted. The time it takes to receive an echo reply is recorded and averaged.

This loop continues until the test duration is reached. Upon exit a report of the minimum, maximum, and average round trip times is printed.

## SITEST

Sitest makes use of many important library functions such as packet construction, address minimization, and socket operations. After any modifications to the library sitest should still compile and run without crashing or exiting with an error message.

The last test that sitest runs measures the rate at which 1024000 packets can be processed. This does not actually transmit any packets but can serve as a quick performance measure to get an idea of the lower bound on the processor and memory access times.

## TESTBENCH

The testbench utility runs an instance of the FCT routing algorithm in a separate high priority thread. The address of the FCT node along with the MMT uid may be specified as arguments. Any number of interfaces can be used for communication using the either the –i option to specifically list the desired interfaces or –I to select all available interfaces. Interfaces may be excluded using the –x option.