Modelnet Howto

David Becker and Ken Yocum <(becker,grant)@cs.duke.edu>

version 0.99

Copyright Notice

Contents

Chapter 1

Modelnet Overview and Requirements

Modelnet software and documentation are beta.

Modelnet *emulates* wide-area network conditions for testing distributed applications, such as peer to peer systems or content distribution networks, within a local area network.

This document explains how to:

- Build and install Modelnet - 'Installing Modelnet' on page 5
- Create a *target* network topology to emulate - 'Generating a Model Network' on page 9
- Deploy the topology on the host cluster - 'Deploying a Model Network' on page 13
- Run an application across Modelnet - 'Running experiments on a Model Network' on page 15

1.1 Overview

Modelnet is designed to run on a machine room cluster to evaluate wide area distriubted systems. One or more of the cluster machines is set aside for traffic emulation, while the remainder can be used to operate as nodes in the application. When these node communicate with each other, those IP packets are sent through the emulators to create the illusion that application packets are crossing the wide area Internet.

To use Modelnet, a virtual network topology must first be created. This virtual network is what the application traffic will experience. The topology contains all the links and nodes of the virtual network and spedifies the link characteristics including bandwidth, queueing, propagation delay, and drop rate. Modelnet includes tools to create these target topologies.

Modelnet emulates a target topology by forwarding all application packets to central network emulation machines. Using the source and destination IP addresses, the emulators determine a path through the virtual topology and handle the packets according to that path. Each hop on this path has certain bandwidth, queueing, propagation delay, and drop characteristics. This hop-by-hop emulation subjects the IP traffic to realistic wide area effects including congestion. The packet emulation work all occurs in real time with millisecond accuracy.

Application hosts are configured with IP aliases on a private subnet (typically the 10.0.0.0/8 network) dedicated to Modelnet emulation. These end hosts send packets over the emulation subnet to the central emulation machines. Applications make their IP traffic go through Modelnet simply by using the IP addresses from the emulation subnet's address space.

For many distributed systems and virtual networks, a typical cluster machine has far more CPU power and network bandwidth than a single instance of the application requires. Modelnet takes advantage of this by creating, possibly, hundreds of virtual nodes on each application host. The application machines have an IP alias (from the emulation subnet) for each virtual node it hosts. Modelnet provides a dynamic library to force all application packets to go out to an emulator, even if they are addressed to another virtual node on that same host. Modelnet includes tools to assist executing applications on large numbers of virtual nodes.

1.2 Cluster requirements

A Modelnet cluster can be as small as two machines; one emulator running FreeBSD, and one machine hosting the virtual nodes. We have successfully hosted virtual nodes on Linux, XenLinux, Solaris and FreeBSD. The crucial feature required to host a virtual node is IP aliasing. With IP aliases, entries can be added to the route table so the host properly handles Modelnet IP packets.

In practice, at Duke we use Linux Debian machines to host virtual nodes, and so this distribution has the most support for that system in terms of scripts and automation. However we have also used ModelNet with Fedora Core 2 distributions, and include notes on the proper packages for that installation.

A LAN with gigabit ethernet links to the emulators gets the best utilization. A gigahertz or faster CPU with a gigabit ethernet NIC on a 64/66MHz PCI slot is a good match to get the most traffic through each emulator.

1.3 Downloads and Related Packages

- Modelnet (`http://issg.cs.duke.edu/modelnet`) - the current release page for Modelnet.

- FreeBSD (`http://www.FreeBSD.org`) - the emulator nodes must run 4.5-RELEASE or newer 4.x kernel. Also, for fine grain timing accuracy, a custom FreeBSD kernel is required to change the clock hertz to 10KHz. Here are the 4.8 ISO images (`ftp://ftp.FreeBSD.org/pub/FreeBSD/ISO-IMAGES-i386/4.8`).

- Boost Graph Library (`http://www.boost.org/libs/graph/doc`)

- Xerces XML Parser (http://xml.apache.org/xerces-c/) for C++.
- Perl modules
  - Graph 0.20105 (http://search.cpan.org/dist/Graph/)
  - Heap (http://search.cpan.org/author/JMM/Heap-0.50/)
  - XML::Simple (http://search.cpan.org/author/GRANTM/XML-Simple-2.07/lib/XML/Simple.pm)

ModelNet requires version 0.2xxxx of the Perl Graph library. See the CPAN site link in the previous section for downloading the right Graph package. The system should be able to build the user-level tools against version 1_32 of the boost library on FreeBSD 4.10, Debian, and Redhat 9.0.

1.3.1 FreeBSD support

For FreeBSD hosts, the package ftp page (ftp://ftp5.freebsd.org/pub/FreeBSD/ports/packages/All) has the latest perl modules and gexec dependencies. Set the `PACKAGESITE` environment variable to this URL (trailing / is significant)

```
ftp://ftp.freebsd.org/pub/FreeBSD/ports/packages/Latest/
```

To download,

```
pkg_add -r p5-XML-Simple linuxthreads libgnugetopt boost xerces-c2
```

You will need to install the p5-Graph (version 0.20105) package off of the CPAN web site.

1.3.2 Debian support

For Debian hosts, the current testing release(sarge) has the boost and XML libraries. apt-get these packages: `libxerces23-dev libboost-graph-dev libxml-simple-perl libssl-dev` For the woody(stable) version of Debian, libxerces21-dev is available via apt-get in the Modelnet debian download tree.

1.3.3 Fedora Core 2 support

For Fedora hosts, you will need to install the following package versions: `xerces-c-devel-2.5.0-1.n0i.1 openssl-0.9.7a-35 boost-devel-1.31.0-7 perl-XML-Simple-2.12-1.1 perl-Graph-0.20105`

Openssl, boost, perl-XML-Simple, can be found using yum. However Xerces and Graph can be installed separately from these sites:

- http://ftp.iasi.rdsnet.ro/mirrors/reb00t.com/fedora-2/RPMS
- http://rpmfind.net/linux/RPM/dag/fedora/1/i386/perl-Graph-0.20105-1.1.fc1.rf.noarch.html

1.3.4 gexec

For large scale remote root execution, Modelnet is designed to use gexec and sudo. See 'Build and Install gexec' on page 7 and 'Running Programs on a Virtual Node' on page 15.

- gexec (`http://www.theether.org/gexec`) | authd (`http://www.theether.org/authd`) | libe (`http://www.theether.org/libe`)
- Modelnet gexec patch (`http://issg.cs.duke.edu/modelnet/gexec-0.3.5-1`)
- OpenSSL (`http://www.openssl.org/`)

Chapter 2

Installing Modelnet

2.1 Binary packages

Modelnet binary packages are available for Debian and FreeBSD systems. For Debian woody (stable), add these lines to /etc/apt/sources.list:

```
deb http://issg.cs.duke.edu/modelnet/debian woody main
deb-src http://issg.cs.duke.edu/modelnet/debian woody main
```

For Debian sarge (testing), add these lines to /etc/apt/sources.list:

```
deb http://issg.cs.duke.edu/modelnet/debian sarge main
deb-src http://issg.cs.duke.edu/modelnet/debian sarge main
```

and install with `apt-get`:

```
apt-get install modelnet
```

This will pull in on the related libraries and perl modules from the regular Debian distribution sites.

For FreeBSD, a kernel modification is strongly recommended (see 'Custom FreeBSD kernel' on the following page). Also the prerequistie packages must be downloaded first (see 'FreeBSD support' on page 3). To download the Modelnet package set the `PACKAGESITE` environment variable to this URL (trailing / is significant)

```
ftp://ftp.cs.duke.edu/pub/modelnet/FreeBSD/
```

To download,

```
pkg_add -r authd gexec modelnet
```

2.2 Building Modelnet from source

Unpack the distribution with tar -

```
tar xfz modelnet-0.0.tar.gz
```

On Linux and FreeBSD, configure and build in your OS-specific object directories -

```
cd modelnet-0.0
```

For Linux:

```
mkdir linux
cd linux
../configure
gmake
```

On FreeBSD, the `modelnet.ko` module is already built and included in the distribution file. However if you do want to build the kernel module from source then you need to tell configure where your kernel sources are. If you want to use the included .ko, just run configure with no options.

```
mkdir freebsd
cd freebsd
../configure --with-fbsdsys=/freebsd/4.XX/src/sys/
gmake
```

2.3 Installing Modelnet from source

Since modelnet has components that need to execute on several different OSs, its easiest to install to a local disk. The default prefix set by configure is /usr/local, so this would happen by default. An alternative is to configure with a prefix in nfs space. Either way, the prefixes must be consistent across all the hosts for the remote execution scripts to operate successfully.

```
gmake install
```

2.4 Custom FreeBSD kernel

For the best fidelity in the emulated network links, the FreeBSD kernel on the emulators needs the clock rate set to 10000 Hertz. In FreeBSD, the HZ parameter is a config time parameter so you have to build a kernel to change it from the default of 100Hz.

See the FreeBSD Handbook (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig.html) chapter on configuring kernels for full details of configuring, building and installing a FreeBSD kernel.

To set HZ, add a line to the conf file in `sys/i386/conf` that says

```
options HZ=10000
```

Briefly, to build a kernel run `config`*conf_file*in the conf dir, then `make` in the sys/compile dir for that configure, and finally copy `kernel` to / and reboot.

2.5 Build and Install gexec

Unpack gexec and patch it. The Modelnet patch allows the gexec system to on FreeBSD hosts.

```
tar xfz libe-0.2.2.tar.gz
tar xfz authd-0.2.1.tar.gz
tar xfz gexec-0.3.5.tar.gz
patch -p0 < modelnet.patch
```

libe needs to be built both on a linux host and freebsd host and installed only on the build hosts.

```
cd libe-0.2.2
mkdir linux
cd linux
../configure
make
make install
```

authd needs to be built and configure both on a linux host and a freebsd host. It needs to be configured and run on all hosts in the emulation cluster.

```
cd authd-0.2.1
mkdir linux
cd linux
../configure
make
make install
```

Create and distribute a cluster key pair as describe on the authd page (http://www.theether.org/authd/). Then authd can be started on all nodes. It is meant to be run at boottime and includes a redhat style startup script that goes in /etc/init.d.

gexecd needs to be built both on a linux host and a freebsd host. It needs to be run on all hosts in the emulation cluster. gexecd can be started from inetd or run stand-alone at startup.

```
cd gexec-0.3.5
mkdir linux
cd linux
../configure
make
make install
```

The `deploy` command (‘Automated Deployment’ on page 14) relies on all hosts in the cluster running gexecd and authd.

Chapter 3

Generating a Model Network

To run a modelnet network, you must create several XML files:

- graph - lists the nodes and links of the virtual network
- route - contains route data for paths through the virtual network
- machines - lists the machines that can be emulators or host virtual nodes.
- model - matches nodes and links to host machines and emulator machines

Modelnet operation requires the route and model file. The graph and machines files are used to create the route and model files. Perl tools are included to create these files, and these tools can be modified to suite particular objectives.

3.1 Creating the graph and route files

The first step to using Modelnet is to create a target network topology to emulate. The topology can be created by generation tools, such as Inet (`http://topology.eecs.umich.edu/inet/`), BRITE (`http://cs-www.bu.edu/faculty/matta/Research/BRITE`), or GT-ITM (`http://www.cc.gatech.edu/projects/gtitm`), or from actual network measurements, such as the RON data (`http://nms.lcs.mit.edu/ron/data/`) from MIT. The current version of Modelnet directly supports the Inet tool. Many situations call for custom topologies. In that case, the perl tools can be modified to generate a graph file according to whatever virtual network topology is required.

3.1.1 Using inet2xml to generate graph file

Modelnet includes a perl tool to convert Inet output to a graph file, and set the link parameters for all the edges in the virtual network topology. This example creates a network of 4000 nodes plus 100 clients attached among 25 stubs spread throughout the toplogy. The links will be given default characteristics.

```
inet -n 4000 | inet2xml -p 100 among 25 stubs > example.graph
```

`inet2xml` can assign values for bandwidth, latency, and drop rate, for four type of links (GT-ITM style link names): 'client-stub', 'stub-stub', 'stub-transit', 'transit-transit'. Link paramter definitions:

- bandwidth - kilobits per second
- latency - milliseconds of delay per packet. Can be infered from length of link in those topologies that specify node coordinates.
- drop rate - fraction of packets that are dropped. Does not include those dropped due to queue overflows.

Node type definitions:

- transit node - node in the virtual network topology corresponding to a router in the wide area internet.
- stub node - a gateway for client nodes to the access the network
- client node - edge node in the virtual network corresponding to an computer attached to the wide area internet
- virtual node - sysnonymous with client node. Each VN is assigned an IP address in the modeled network. Modelnet emulates the end-to-end traffic between VNs.

Link type definitions:

- client-stub - connects a virtual node to an interior node at the edge of the virtual network. Corresponds to a link from a computer to a gateway router.
- stub-stub - connects two nodes at the edge of the virtual network
- stub-transit - connects a stub node to transit node. Correspond to a link from a site (an AS) to a backbone network.
- transit-transit - corresponds to links within or between backbone networks.

With `inet2xml`, you can specify all the link parameters for all the links types, or give a min-max range where inet2xml randomly picks a value.

```
usage: inet2xml [-l] [-q qcnt] -p <vncnt> among <stubcnt> stubs
 [<link type> <kbps bw> <ms delay> <drop fraction>]+ |
 [min-<link type> <kbps bw> <ms delay> <drop fraction>
  max-<link type> <kbps bw> <ms delay> <drop fraction>]
```

```
-l    Set delay based on link lengths derived from inet node location
-q qcnt
      Queue length for on all links. Default is  10
-p <vncnt> among <stubcnt> stubs
      Create <vncnt> virtual nodes spread among <stubcnt> stubs
<link type>
      Types can be: client-stub stub-stub stub-transit transit-transit
<kbps bw>
      integer kilobits per second
<ms delay>
      integer milliseconds of link latency
<drop fraction>
      real value from fraction of packets dropped
```

3.1.2 XML graph file format

The graph file has 3 subsections: vertices, edges and specs. For this example, take a simple graph that has one link connecting two nodes, then attach one client node to one node and two clients to the other node. This graph then has two stub nodes, three clients nodes, and 8 uni-directional edges connecting them. The resulting XML graph file is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<topology>
  <vertices>
    <vertex int_idx="0" role="gateway" />
    <vertex int_idx="1" role="gateway" />
    <vertex int_idx="2" role="virtnode" int_vn="0" />
    <vertex int_idx="3" role="virtnode" int_vn="1" />
    <vertex int_idx="4" role="virtnode" int_vn="2" />
  </vertices>
  <edges>
    <edge int_dst="1" int_src="2" int_idx="0" specs="client-stub" int_delayms
    <edge int_dst="2" int_src="1" int_idx="1" specs="client-stub" dbl_kbps="7
    <edge int_dst="1" int_src="3" int_idx="2" specs="client-stub" />
    <edge int_dst="3" int_src="1" int_idx="3" specs="client-stub" />
    <edge int_dst="0" int_src="4" int_idx="4" specs="client-stub" />
    <edge int_dst="4" int_src="0" int_idx="5" specs="client-stub" />
    <edge int_dst="1" dbl_len="1" int_src="0" int_idx="0" specs="stub-stub" /
    <edge int_dst="0" dbl_len="1" int_src="1" int_idx="1" specs="stub-stub" /
  </edges>
  <specs >
    <client-stub dbl_plr="0" dbl_kbps="64" int_delayms="100" int_qlen="10" />
    <stub-stub dbl_plr="0" dbl_kbps="1000" int_delayms="20" int_qlen="10" />
  </specs>
```

```
</topology>
```

The <specs> section of each <edge> can be overridden on a per-edge basis. Any link parameter expressly stated as an <edge> attribute overrides the value in the associated <specs>. For example, edge 0 overrides the delay to be 1ms and edge 1 overrides the bandwidth to be 768 Kbit/s.

3.1.3 Creating the route file from the graph file

The route file store the shortest paths across the virtual network for all pairs of virtual nodes.

```
allpairs example.graph > example.route
```

3.2 Creating the machines and model files

The machines file is written by hand to list the machines available to be emulators or host virtual nodes. This example is for cluster of 3 machines name larry, curly and moe and designates them as an emulator and two hosts.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<hardware>
<emul hostname="larry"/>
<host hostname="curly"/>
<host hostname="moe"/>
</hardware>
```

The model file is create by the mkmodel perl script. mkmodel assigns the virtual nodes to hosts, and assigns links to emulators.

```
mkmodel example.graph example.machines > example.model
```

Chapter 4

Deploying a Model Network

Once the model and route files are created, the emulation cluster can be configured to be the virtual network. Root access on all machines is required for this operation. With a handful of machines, it is feasible to manually setup Modelnet.

Modelnet requires all machines in the emulation cluster have the XML::Simple (http://search.cpan.org/author/GRANTM/XML-Simple-2.07/lib/XML/Simple.pm) perl module installed.

4.1 Sudo support

Root access is required to deploy a network topology. Using sudo, this access can be fine tuned and password-free. The deploy scripts assume sudo is configured on each machine used for Modelnet.

For controlled access, the /etc/sudoers (or /usr/local/etc/sudoers on FreeBSD) can specify the Modelnet commands. Add these lines to sudoers, for password-free access:

```
Host_Alias      MODELHOSTS = larry moe curly
User_Alias      MODELERS =  login names go here
Cmnd_Alias MODELCMDS = /sbin/ifconfig, /sbin/route, \
        /sbin/kldload, /sbin/kldunload, /sbin/ipfw, /sbin/sysctl, \
 /usr/local/bin/modelload, /usr/local/bin/ipfwload
MODELERS MODELHOSTS = NOPASSWD: MODELCMDS
```

To test your sudo access, do `sudo -l`

4.2 Manual Deployment

Modelnet can be deployed by logging in to each host, and running the `deployhost` command. This commond configures all the virtual IP addresses, routes, and loads the topology into the emulator.

```
deployhost example.model example.route
```

Once `deployhost` has run on every machine in the hosts file, the system is ready to emulate the virtual network.

4.3 Automated Deployment

For large emulations, deployment can be automated as a a single command when gexec remote access to the cluster is available. See ‘Build and Install gexec’ on page 7.

```
deploy example.model example.route
```

The `deploy` can be run on any node in the `gexec` cluster.

Chapter 5

Running experiments on a Model Network

5.1 Running Programs on a Virtual Node

Once a network topology is deployed onto the emulation hardware, the system is live and it can test a distributed application. All packets an edge node transmits with a source and destination IP in the 10.0.0.0/8 network will be sent its emulator. This requires that the src and dst IP be set correctly. Modelnet provides `libipaddr` to control the IP addresses applications use.

5.2 vnrun

`vnrun` will execute a program on a virtual node, or on all virtual nodes. It creates an environment that make sure the source address is set correctly for any internet sockets. The example runs the date program on all virtual nodes listed in example.model.

```
vnrun all example.model gnutella
```

A copy of `gnutella` is run on every virtual node. If the `gnutella` program opens a socket, the IP source address will be set according the virtual running that instance of gnutella.

`vnrun` can also start a single instance of an application if a virtuan node number is given instead of "all".

```
usage: vnrun [-d] < VN# | all > <file.model> <command>
```

5.3 libipaddr

libipaddr has only been tested on linux glibc-2.2.5

`vnrun` uses `libipaddr` to make applications use the correct IP addresses. `libipaddr` is a shared library so it requires that applications are dynamically linked executables. For Unix systems, it relies on the `LD_PRELOAD` of `ld.so` to interpose its own version of socket related system calls. Typical applications do not explicitly set the source IP address. The libipaddr versions of `bind(2)`, `sendto(2)` and other syscalls explicitly set the source and destination IP addresses.

The `libipaddr` does not work with static binaries or with RAW sockets. In particular, *ping* uses RAW sockets and therefore cannot have its addresses manipulated by `libipaddr`. Test connections with netperf or some other higher level application than `ping(8)`.

`libipaddr` sets the source address to be the address in the environment variable SRCIP. Using sh syntax:

```
$ LD_PRELOAD=prefix/lib/libipaddr.so SRCIP=10.0.0.1 netperf -H 10.0.1.1
```

The `netperf` packets transmitted with have a source IP of 10.0.0.1. Server processes, eg. `netserver`, address return packets simply by swapping the source and destination addresses. This means pure server processes do not need the libipaddr system calls.

Packets being sent between two virtual nodes on the same edge node must be prevented from going through the loopback interface. This is why the destination IP address is also set by `libipaddr`. This forces packets to actually go to the emulator when the destination virtual node happens to reside on the same host as source virtual node. This is done by turning on bit 23 of the destination address. Since the 10.128.0.0/24 subnet does not reside on the host with the 10.0.0.0/24 net, it will send the packet to the emulator all other packets for 10.0.0.0/8. In this example, even though 10.0.0.1 and 10.0.0.2 are on the same host, the netperf packets in both directions will go through the emulator

```
$ LD_PRELOAD=prefix/lib/libipaddr.so SRCIP=10.0.0.1 netperf -H 10.0.0.2
```

The emulator turn off bit 23 of the destination IP. If 23 was set, it will turn it on in the source IP address before delivering the packet to the edge node with the destination virtual node. This is so servers will automatically send replies with bit 23 set. So again, pure servers do not need `libipaddr` PRELOAD-ed. This feature is the default behavior of `libipaddr`. To disable this feature, define the environment variable `KEEP_DSTIP`.

Chapter 6

Validating and Troubleshooting Modelnet

The configuration for Modelnet itself, and the experimental distributed applications that use it, are complex systems that must be validated to confirm it is all working as expected.

netperf (`http://www.netperf.org/`) is the handiest tool to validate link performance. Use it with libipaddr ('libipaddr' on the facing page)to confirm link bandwidth of the virtual topology with both TCP and UDP packets.

Check mbuf limits, `netstat -m`, on forwarders. Check `systat -ip`, on forwarders for packet throughput.