Simulation of IPv6 Networks with OMNeT++

Slide 1

Y. Ahmet Şekercioğlu

Centre for Telecommunication and Information Engineering
Monash University, Melbourne, Australia

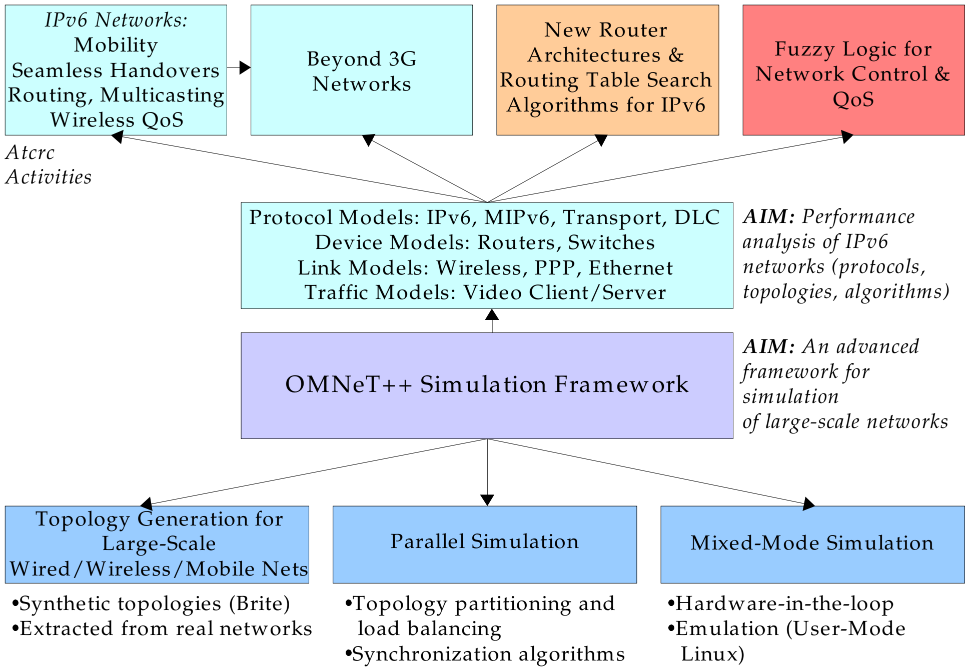
Contents

Slide 2

- OMNeT++ discrete-event simulation framework and our simulation research activities
- An overview of IPv4/IPv6 simulation models and IPv6Suite capabilities
  - Example 1 "Ethernet Network": IPv6 packets, autoconfiguration and duplicate address detection (DAD)
  - Example 2 "ICMPv6 Echo Request/Reply": Observing data while simulation is running
- Modeling and Simulating Packet-Switching Networks
  - An introduction to simulation of packet-switching networks: *Stop and Wait Protocol*
  - Experiments with a *simple* packet switch
- Learning to write your own IPv6 network simulation models
  - First IPv6 Model: Two clients, an Ethernet hub and a video server
- Simulation of Mobile IPv6 networks
  - Example 3: A Wireless LAN with four access points

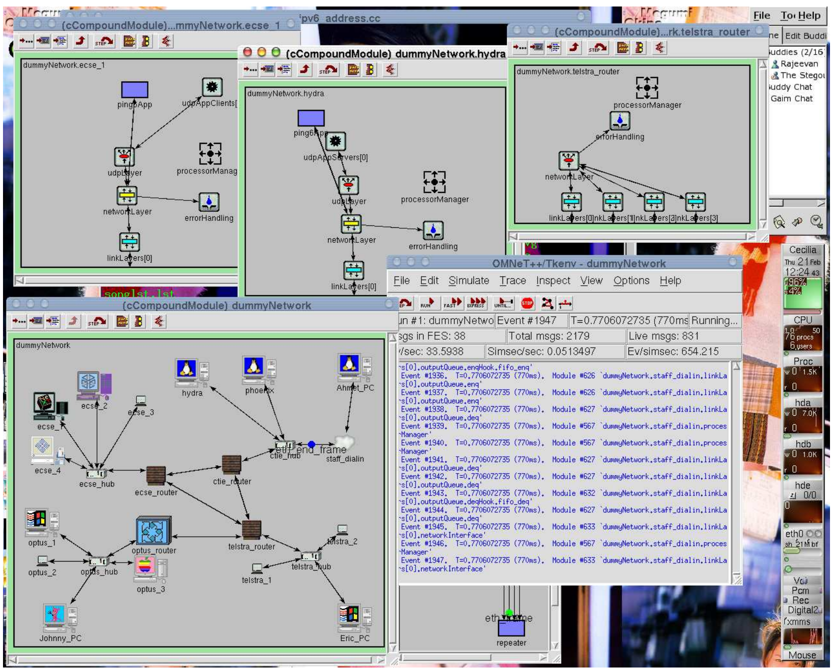
Slide 3

Simulation Activities



Slide 4

Simulation Framework: OMNeT++



Why OMNeT++?

Slide 5

- An open source framework [OMN96] (free for research and educational purposes, GPL-like license) which has an active, cooperating research community.
- Runs on Windows NT, Windows 2000, Windows XP, Linux, Compaq Tru64 and Sun Solaris operating systems.
- Fully object-oriented architecture. Models can be formed by assembling other models (“compound modules”), and dynamic behaviour is modeled as C++ based modules (“simple modules”).
- Topology is defined through text files (excellent for automatic topology generation).
- Can simulate large networks - good scalability.
- Dynamic graphical user interface for tracing packet flows (very handy for fault-finding and debugging models).

A Sample of Commercial Companies Using OMNeT++

Slide 6

- **Native Networks** is a manufacturer of high speed connectivity solutions for optical access networks. They use OMNeT++ for simulating their systems' performance in terms of bandwidth utilization and teletraffic delay. They also use OMNeT++ for examining the behaviour of our hardware's state-machines and to generate verification vectors.
- **Meriton Networks** is using OMNeT++ to develop a performance model for a network of their switches, using IETF draft definitions of protocols.
- **American Automobile Association Response Services Center** is using OMNeT++ for evaluating and predicting network and processor performance. They have selected OMNeT++ to model their internal network to support load projections.
- **Wipro Technologies** is using OMNeT++ for network traffic modeling of ring and shared LAN topologies to analyze the behaviour of layer-2 protocols.

Slide 7

Contents

- OMNeT++ discrete-event simulation framework and our simulation research activities
- An overview of IPv4/IPv6 simulation models and IPv6Suite capabilities
  - Example 1 "Ethernet Network": IPv6 packets, autoconfiguration and duplicate address detection (DAD)
  - Example 2 "ICMPv6 Echo Request/Reply": Observing data while simulation is running
- Modeling and Simulating Packet-Switching Networks
  - An introduction to simulation of packet-switching networks: *Stop and Wait Protocol*
  - Experiments with a *simple* packet switch
- Learning to write your own IPv6 network simulation models
  - First IPv6 Model: Two clients, an Ethernet hub and a video server
- Simulation of Mobile IPv6 networks
  - Example 3: A Wireless LAN with four access points

Slide 8

An Overview of OMNeT++ IPv4/IPv6 Simulation Model Sets

	EXISTING MODELS	NEEDED MODELS
Application Layer	Video client-server	Voice, Web, file transfer
Transport Layer	UDP, TCP	TCP testing only done with IPv4 models, IPv6 waiting.
Network Layer	IPv4, IPv6	Routing protocols, Diff-Serv, Queue management
Data Link Control Layer	Ethernet, PPP, IEEE 802.11	MAC for cellular
Physical Layer	Simple access	Radio propagation models, mobility models

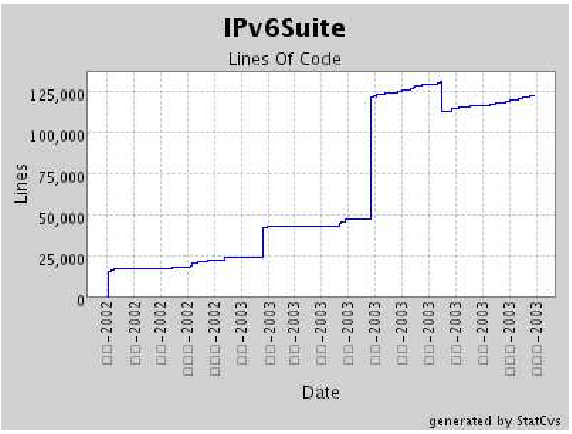
IPv6Suite

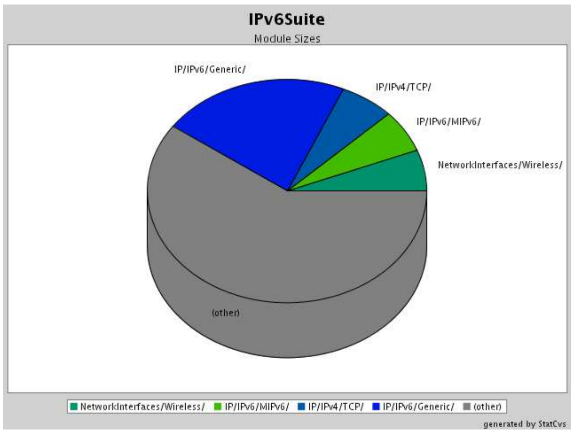
Slide 9

- We have developed a set of OMNeT++ models for accurate simulation of IPv6 protocols.
- Our simulation set models the functionality of the following RFCs:
  - RFC 2373 *IP Version 6 Addressing Architecture*
  - RFC 2460 *Internet Protocol, Version 6 (IPv6) Specification*
  - RFC 2461 *Neighbor Discovery for IP Version 6 (IPv6)*
  - RFC 2462 IPv6 *Stateless Address Autoconfiguration*
  - RFC 2463 *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*
  - RFC 2472 *IP Version 6 over PPP*
  - RFC 2473 *Generic Packet Tunneling in IPv6*
  - *RFC draft:* `draft-mobile-IPv6-spec`
- We have developed XML based parsing modules for flexible configuration of network node parameters (more details later)

IPv6Suite is a Large Software Project

Slide 10

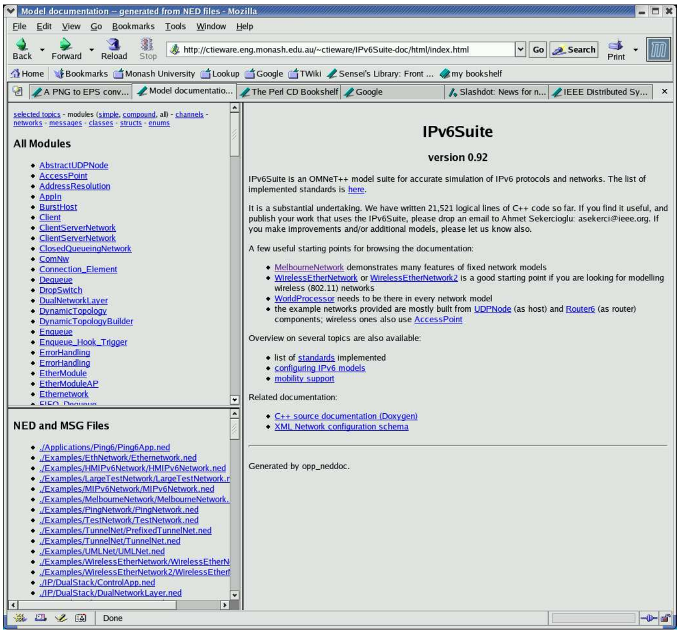




The Team: E. Wu, J. Lai, S. Woon, J. Fjeldberg, A. Şekercioğlu

IPv6Suite Web Site for Downloads and Documentation

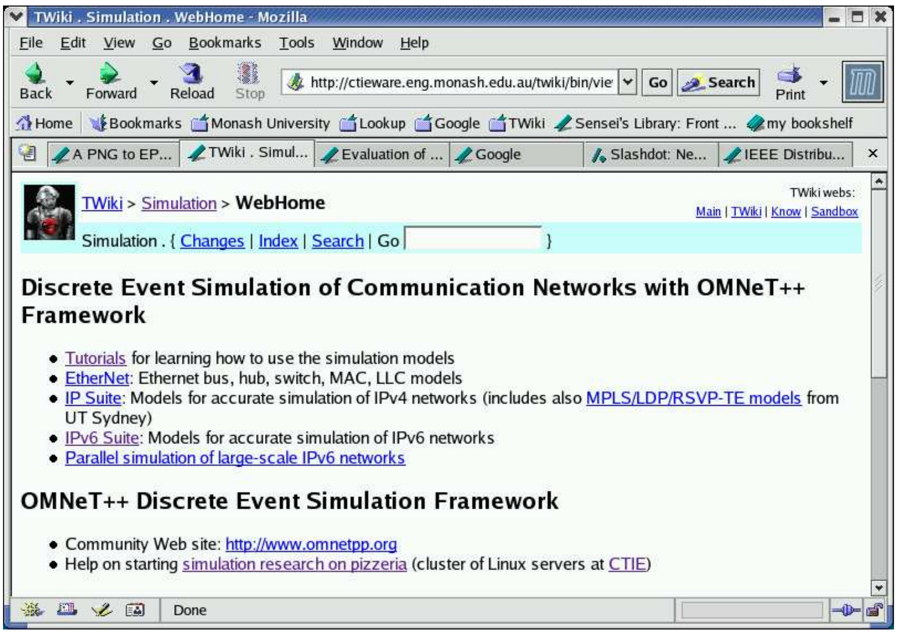
Slide 11



http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/IPv6Suite

Related Simulation Research at CTIE

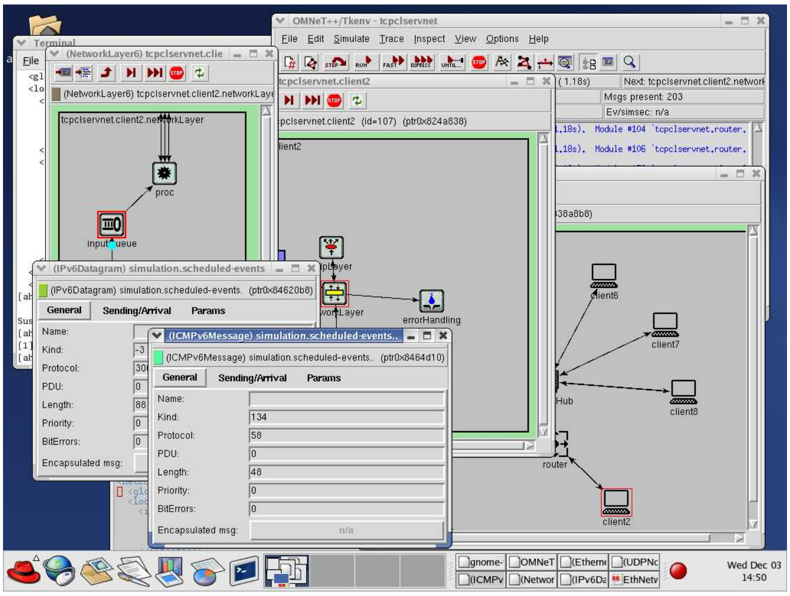
Slide 12



http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation

Example 1: Address Auto-Configuration, Duplicate Address Detection in IPv6

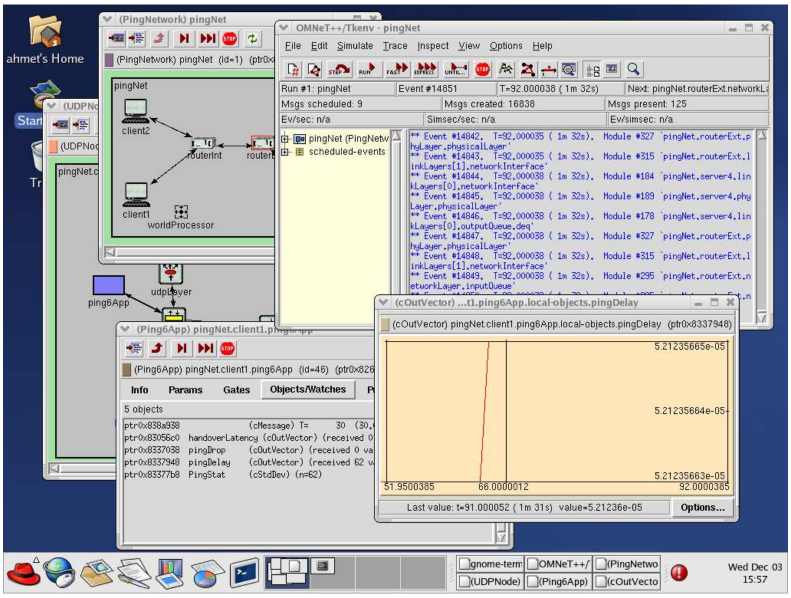
Slide 13



- Can be found among the examples downloadable through the IPv6Suite Web site (`EthNetwork`).
- `IPv6Headers.h` and `ICMPv6Message.h` define the packet types.

Example 2: ICMPv6 Echo Request/Reply - Observing Network Dynamics

Slide 14



- Can be found among the examples downloadable through the IPv6Suite Web site (`PingNetwork`).
- We can observe the dynamic values while the simulation is in progress.

Contents

Modeling and Simulating Packet-Switching Networks

Slide 16

- Step 1: Identify the components and their decomposition: compound modules Vs simple modules.
- Step 2: Create the compound modules by using NED and/or GNED.
- Step 3: Create the simple modules ("dynamic behaviour") by writing the C++ code.
- Step 4: Create a series of experiments ("runs") by writing the `omnetpp.ini`.
- Step 5: Create the simulation executable.
- Step 6: Run the experiments, collect and analyze the data (PLOVE, custom scripts, other data visualization and analysis tools).

Introduction to Simulation of Packet-Switching Networks: Stop-And-Wait

Slide 17

- We will simulate two hosts (a sender and a receiver) connected via a “perfect” communication link.
- Sender will send a “data packet” to the receiver, and will wait for an “acknowledgment packet”.
- We need a project directory, I suggest `$HOME/oppsim/stopnwait`.
- Then, let’s write the topology file: `stopnwait.ned`.

stopnwait.ned

Slide 18

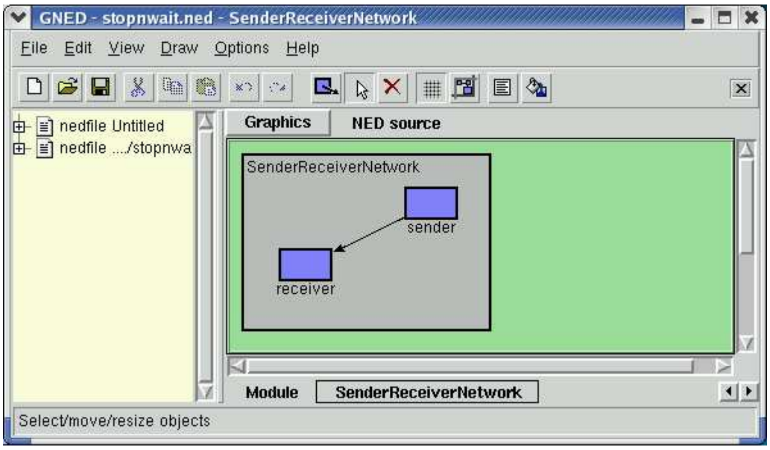
```
simple Host
    gates:
        in: in;
        out: out;
endsimple

module SenderReceiverNetwork
    submodules:
        sender: Host;
        receiver: Host;
    connections:
        sender.out --> receiver.in;
        sender.in <-- receiver.out;
endmodule

network // We can have multiple networks defined in a single NED file.
    senderreceivernet : SenderReceiverNetwork
endnetwork
```

GNED: Graphical Editor for NED Topology Files

Slide 19



We can use GNED to view and edit the topology: `gned stopnwait.ned`.

Implementation of the `Host` Dynamic Behaviour

Slide 20

- We now need to implement the functionality of the simple module `Host`.
- This is achieved by writing two C++ files:
  - `host.h` → Class definition of the host
  - `host.cc` → Implementation of the host

Slide 21

host.h

```
#include "omnetpp.h"

// Derive the Host class from cSimpleModule.
class Host : public cSimpleModule
{
  // This is a macro; it expands to constructor definition etc.
  // 16384 is the size for the coroutine stack (in bytes).
  Module_Class_Members(Host, cSimpleModule, 16384);

  // This redefined virtual function holds the algorithm.
  virtual void activity();
};
```

Slide 22

host.cc - Part 1/2

```
#include <stdio.h>
#include <string.h>
#include "omnetpp.h"
#include "host.h"

Define_Module(Host); // register the module types to the OMNeT++

void Host::activity()
{
  ev << "Hello World! I'm " << name() << ".\n";

  // Am I sender or receiver?
  if (strcmp("sender", name()) == 0) {
      // Sender will send the first packet and will wait for ack.
      cMessage *msg = new cMessage(name());
      ev << name() << " sending 1st msg: "<< msg->name() << ".\n";
      send(msg, "out");
  }
```

Slide 23

host.cc - Part 2/2

```
        // Infinite loop to process events.
        for (;;) {
            cMessage *msgin = receive();
            ev << name() << " got msg: " << msgin->name() << ".\n";
            delete msgin;
            wait(1.0);
            cMessage *msg = new cMessage(name());
            ev << name() << " sending msg: " << msg->name() << ".\n";
            send(msg, "out");
        }
    }
```

Slide 24

omnetpp.ini

We now write the omnetpp.ini file which tells simulation system what to do.

```
[General]
ini-warnings = no

[Tkenv]
default-run=1

[Cmdenv]
module-messages = yes
verbose-simulation = no

[Run 1]
network=senderreceivernet

# I could define a series of experiments as [Run 2] ...
```

Time to Run the Simulation

Slide 25

- We create the `Makefile` which will help us compile and link our program to create the executable `stopnwait`:

```
opp_makemake
```

- Let's compile and link:

```
make
```

- Let's run:

```
./stopnwait
```

Models of Communication Links

Slide 26

- In real life communication networks, the links carrying the packets involve propagation delays, bit error rates and varying transmission capacities.
- OMNeT++ allows researchers to have sophisticated communication link models.
- As an example we can modify the `stopnwait.ned` to have a more realistic link:

```
channel srlink
    delay 0.5 //sec.
    datarate 100000
end channel
...
sender.out --> srlink --> receiver.in;
sender.in  <-- srlink <-- receiver.out;
```

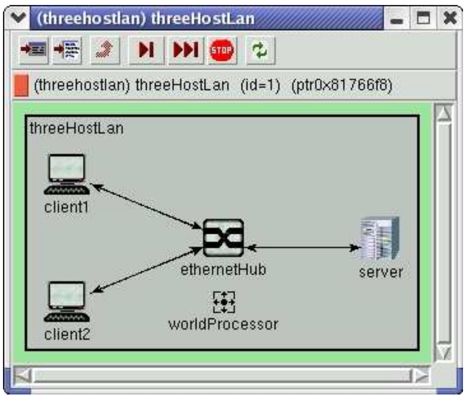
Slide 27

Contents

- OMNeT++ discrete-event simulation framework and our simulation research activities
- An overview of IPv4/IPv6 simulation models and IPv6Suite capabilities
  - Example 1 "Ethernet Network": IPv6 packets, autoconfiguration and duplicate address detection (DAD)
  - Example 2 "ICMPv6 Echo Request/Reply": Observing data while simulation is running
- Modeling and Simulating Packet-Switching Networks
  - An introduction to simulation of packet-switching networks: *Stop and Wait Protocol*
  - Experiments with a *simple* packet switch
- Learning to write your own IPv6 network simulation models
  - First IPv6 Model: Two clients, an Ethernet hub and a video server
- Simulation of Mobile IPv6 networks
  - Example 3: A Wireless LAN with four access points

Slide 28

First IPv6 Network Model: `3HostLAN`



We will now build this network by using the `IPv6Suite`. To do this, we will create a project directory `IPv6Suite/Examples/3HostLAN`, and write four files (we will not be writing any CC files since we will be using the existing models):

1. Topology description file `threehostlan.ned`
2. Simulation control file `omnetpp.ini`
3. Network address and routing configuration file `3HostLAN.xml`
4. Compilation configuration file `CMakeLists.txt`

`IPv6Suite` introduces the `.xml` and `CMakeLists.txt` files for fine-grain configuration of the network parameters and flexible control of the complex build process respectively.

`3HostLAN` Topology Description File `threehostlan.ned`

```
import                                          // IPv6Suite modules.
    "EtherHub",
    "UDPNode",
    "WorldProcessor";

module  threehostlan
    submodules:
        worldProcessor: WorldProcessor;         // Does the XML parsing,
                                                // needs to
                                                // be the first in the
                                                // modules list since every
                                                // other module relies on
                                                // this one.
            display: "b=17,17;p=165,156;i=bwgen_s";
        client1: UDPNode;
            parameters:
                numOfPorts = 1;
            gatesizes:
                in[1],
                out[1];
            display: "p=42,56;b=36,32;i=comp";
        client2: UDPNode;
            parameters:
                numOfPorts = 1;
            gatesizes:
                in[1],                          // NED can have an array of
                out[1];                         // gates. See hcube example.
            display: "p=42,156;b=36,32;i=comp";
        server: UDPNode;
            parameters:
                numOfPorts = 1;
            gatesizes:
                in[1],
                out[1];
            display: "p=288,106;b=36,32;i=comp";
        ethernetHub: Hub;
            parameters:
                numOfPorts = 3;
            gatesizes:
                in[3],
                out[3];
            display: "p=165,106;b=32,30;i=xconn";
    connections:
        client1.in[0] <-- delay 10ms <-- ethernetHub.out[0];
        client1.out[0] --> delay 10ms --> ethernetHub.in[0];

        client2.in[0] <-- delay 10ms <-- ethernetHub.out[1];
        client2.out[0] --> delay 10ms -->ethernetHub.in[1];

        server.in[0] <-- delay 10ms <-- ethernetHub.out[2];
        server.out[0] -->delay 10ms --> ethernetHub.in[2];
    display: "p=10,10;b=308,184";
```

```
endmodule

network
    threeHostLan : threehostlan
endnetwork
```

3HostLAN Network Address and Routing Configuration File: 3HostLAN.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE netconf SYSTEM "../../Etc/netconf2.dtd">
<netconf debugChannel="debug1.log:notice:xmlAddresses:UDPVidStrmSvr:Ethernet:debug">
  <global gHostDupAddrDetectTransmits="2"/>
  <local node="server">
    <interface name="eth0">
      <inet_addr>fe80:0:0:0:606:98ff:fe24:52f5</inet_addr>
    </interface>
  </local>
  <local node="client1">
    <interface name="eth0">
      <inet_addr>fe80:0:0:0:606:98ff:fe24:52f6</inet_addr>
    </interface>
  </local>
  <local node="client2">
    <interface name="eth0">
      <inet_addr>fe80:0:0:0:606:98ff:fe24:52f7</inet_addr>
    </interface>
  </local>
</netconf>
```

3HostLAN Simulation Control file omnetpp.ini

```
[General]
network = threeHostLan

total-stack-kb=7535
ini-warnings = no
warnings = no
sim-time-limit = 101s

[Cmdenv]
module-messages = yes
event-banners=no

[Tkenv]
default-run=1
breakpoints-enabled = no
animation-speed = 1.0

[Parameters]
threeHostLan.client1.ping6App.startTime=30
threeHostLan.client1.ping6App.deadline=100
threeHostLan.client1.ping6App.destination="fe80:0:0:0:606:98ff:fe24:52f5"
threeHostLan.client1.ping6App.interval=0.5s

threeHostLan.client2.numOfUDPClientApps=1
threeHostLan.client2.udpAppClients[0].UDPAppClientName=
                                        "UDPVideoStreamCnt"
threeHostLan.client2.udpAppClients[0].UDPServerAddress=
                                        "fe80:0:0:0:606:98ff:fe24:52f5"
threeHostLan.client2.udpAppClients[0].UDPServerPort=7001
threeHostLan.client2.udpAppClients[0].IPversion=6

threeHostLan.server.numOfUDPServerApps=1
threeHostLan.server.udpAppServers[0].UDPAppServerName="UDPVideoStreamSvr"
threeHostLan.server.udpAppServers[0].IPversion=6
threeHostLan.server.udpAppServers[0].UDPPort=7001

threeHostLan.*.IPv6routingFile ="3HostLAN.xml"

include ../../Etc/default.ini
```

Slide 29

CMake Build System

- With IPv6Suite we have started experimenting with the CMake (www.cmake.org) build system.
- CMake is used to control the software compilation process using simple platform and compiler independent configuration files.
- CMake generates native makefiles and workspaces that can be used in the chosen compiler environment.
- Simple configuration files placed in each source directory (called `CMakeLists.txt` files) are used to generate standard build files (e.g., makefiles on Unix and projects/workspaces in Windows MSVC) which are used in the usual way.
- CMake can compile source code, create libraries, generate wrappers, and build executables in arbitrary combinations.

`3HostLAN` Build Process Control File `CMakeLists.txt`

```
SET( 3HostLAN_ned_includes
${TOPDIR}/Nodes
${TOPDIR}/IP/DualStack
${TOPDIR}/NetworkInterfaces
${TOPDIR}/Transport/TCP
${TOPDIR}/Transport/UDP
${TOPDIR}/Applications/Ping6
${TOPDIR}/IP/IPv4/MAC_LLC
${TOPDIR}/IP/IPv6/Generic/
${TOPDIR}/IP/IPv4/QoS
${TOPDIR}/IP/IPv4/IPProcessing
${TOPDIR}/World
${TOPDIR}/PHY)

CREATE_SIMULATION(3HostLAN 3HostLAN_ned_includes threehostlan)
LINK_OPP_LIBRARIES(tk3HostLAN "${OPP_TKGUILIBRARIES}")
```

Compiling and Running the `3HostLAN` Simulation

After writing the `threehostlan.ned`, `omnetpp.ini`, `3HostLAN.xml`, and `CMakeLists.txt`, we are almost ready to see some results. Still a few more steps are needed:

Slide 30

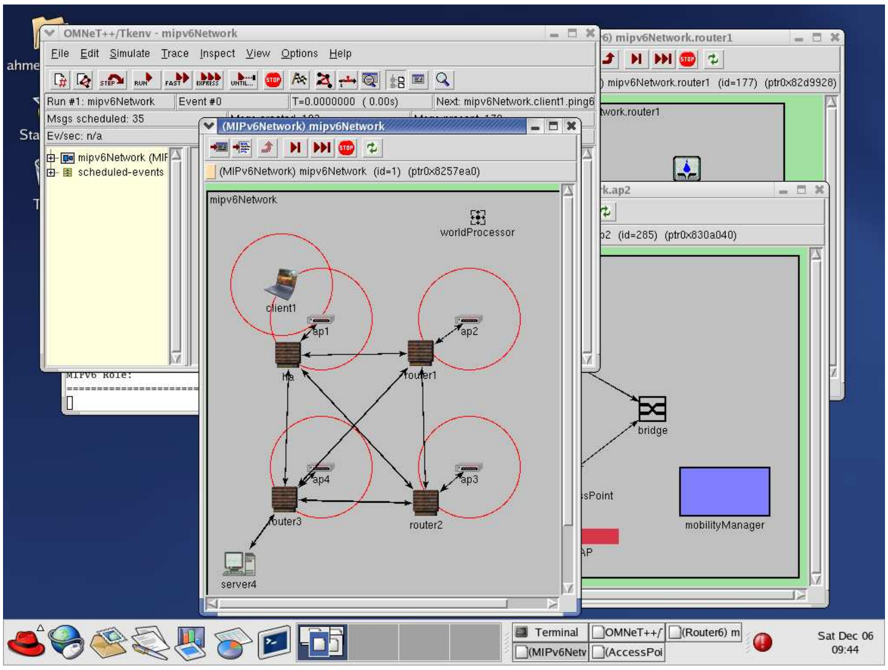
1. We need to include our new model into the build system. To do this, we edit the `/oppsim/IPv6Suite/Examples/CMakeLists.txt` file and add the directory `3HostLAN` to the line
   ```
   SUBDIRS(EthNetwork MelbourneNetwork PingNetwork ....
   ```
   Otherwise, build system will ignore our new model.
2. We go to `/oppsim/IPv6Suite` directory and issue the command
   ```
   cmake .
   ```
   Now CMake will generate the necessary Makefiles.
3. We now go to `3HostLAN` directory and issue the command
   ```
   make
   ```
   to produce the model. From now on if we modify our model, we only need to reissue the `make` command in the `3HostLAN` directory.
4. We run the model: `./3HostLAN`.

Contents

Slide 31

A Wireless LAN with Four Access Points

Slide 32



References

[OMN96] OMNeT++ object-oriented discrete event simulation system. URL reference: http://www.omnetpp.org, 1996.