

# Simulation Study on Large Scale Wireless Networks: Interim Report

Hu Wenzheng Student ID:5050309855

Ma Yinlong Student ID:5050309841

Guo Jie Student ID:5050039278

Lian Tsong Student ID:5050309281

EE department,SJTU

## ***Index Terms*—Large Scale Ad Hoc Networks, NS-2 simulation**

### I. WHAT WE HAVE DONE

**T**hese days we studied the ns-2 node models, and tried to simulate some simulation examples with ns-2.

#### *A. The ns-2 node model*

1) *Basic Components of A Node:* The structure of one unicast node contains two TclObject, one is address-classifier(classifier\_) and the other is port-classifier(dmux\_). These classifiers implement the function of transmitting the packets received to destination agent or link ports.

For a multicast node, it is a little more complicated. Besides of address-classifier and port-classifier, a switch and a multiclassifier are also included.

Here we talk a little about the concept of unicast and multicast.

Unicast is the term used to describe communication where a piece of information is sent from one point to another point. In this case there is just one sender, and one receiver. All LANs (e.g. Ethernet) and IP networks support the unicast transfer mode, and most users are familiar with the standard unicast applications (e.g. http, smtp, ftp and telnet) which employ the TCP transport protocol.

Multicast is the term used to describe communication where a piece of information is sent from one or more points to a set of other points. In this case there is may be one or more senders, and the information is distributed to a set of receivers (their may be no receivers, or any other number of receivers).

Multicasting is the networking technique of delivering the same packet simultaneously to a group of clients. IP multicast provides dynamic many-to-many connectivity between a set of senders (at least 1) and a group of receivers. Since TCP supports only the unicast mode, multicast applications must use the UDP transport protocol.

Every node should at least contains the following parts:

- id or node\_address, the default value is 0;
- neighbor linked table(neighbor\_);

- agent linked table(agent\_);
- nodetype identifier(nodetype\_);
- routing module.

#### *2) Node Method to Config Node:*

##### • **Control Function**

\$node entry: it will return the pointer pointing to the entry of the node

\$node reset: reset all the agent attached to the node

##### • **Address and Port Management**

\$node id: return the index of the node which is automatically arranged when creat node using \$ns node  
\$node agent (port): return the handle, a pointer pointing to the agent whose port number is 'port'. If such an agent doesn't exist, it will return a null string.

\$node add route(s)[destination id][TclObject]:add a new routing path in unicast mode to generate a new classifier. TclObject is the entry of dmux\_, the multi-access port.

##### • **Agent Management**

attach():this function will add the given agent to the agent-linked table(agent\_), allocate a port number to the agent and set its source address.

detach():it is used to remove agent from agent\_,the pointer will point to null agent entry.

##### • **Add Neighbor**

add-neighbor():in each node, the variable neighbor\_ maintain a neighbor table, this function will add a new neighbor into the linked table.

neighbors():this is used to return the neighbor linked table.

3) *Node and Classifier:* In NS2, we can just take Node as a set of classifiers. The simplest unicast node just contain an address-classifier and a port-classifier. When adding more extensive function to the node, more classifiers will be added, for example, multicast node. As more and more function modules are added, it's a must for the node to provide unified interface in order to manage these classifiers.

We will here pay a little attention to the concept and function of classifiers. Classifier is a basic class in NS2 network components. It has two basic derived class, Address-classifier and Multicast-classifier. When receive a packet, the

node is required to check the domain of the packet, which is usually the destination, while sometimes can be source address. Then the node will get a pointer pointed to where the packet will be sent. This process is executed by a classifier object.

The main difference between unicast and multicast classifier is the way how the packets are classified. In unicast classifier, the packets are classified only according to the destination while both source and destination are considered in multicast classifier.

## B. some simulation examples

We tried to simulate some examples on networks. All the models come from `ns/ns-2.28/tcl/ex/` in the `ns-2` directory.

### 1) Simulation on wireless flooding

*1) Initializing in tcl file:* Global Parameters are initialized in `wireless-flooding.tcl`:

```
Mac/Simple set bandwidth_ 1Mb

set MESSAGE_PORT 42
set BROADCAST_ADDR -1
set group_size 4
set num_groups 6
set num_nodes [expr $group_size * $num_groups]
set val(chan) Channel/WirelessChannel ;
# Channel Type
set val(prop) Propagation/TwoRayGround ;
# radio-propagation model
set val(netif) Phy/WirelessPhy ;
# network interface type
set val(mac) Mac/Simple;
# MAC type
set val(ifq) Queue/DropTail/PriQueue ;
# interface queue type
set val(ll) LL;
# link layer type
set val(ant) Antenna/OmniAntenna;
# antenna model
set val(ifqlen) 50;
# max packet in ifq
set val(rp) DumbAgent;
# routing protocol
# size of the topography
set val(x) [expr 120*$group_size + 500]
set val(y) [expr 240*$num_groups + 200]
set ns [new Simulator]

set f [open
    wireless-flooding-$val(rp).tr w]
$ns trace-all $f
set nf [open
    wireless-flooding-$val(rp).nam w]
$ns namtrace-all-wireless
$nf $val(x) $val(y)
```

```
$ns use-newtrace
```

```
# set up topography object
set topo [new Topography]
```

```
$topo load_flatgrid $val(x) $val(y)
```

Attention: The MAC layer here uses `Mac_Simple`, instead of `802.11`; and the routing protocol is `DumbAgent`. Then the parameters of the nodes:

```
#
# Create God
#
create-god $num_nodes

set chan_1_ [new $val(chan)]

$ns node-config -adhocRouting $val(rp) \
    -llType $val(ll) \
    -macType $val(mac) \
    -ifqType $val(ifq) \
    -ifqLen $val(ifqlen) \
    -antType $val(ant) \
    -propType $val(prop) \
    -phyType $val(netif) \
    -topoInstance $topo \
    -agentTrace ON \
    -routerTrace OFF \
    -macTrace ON \
    -movementTrace OFF \
    -channel $chan_1_
```

Each agent keeps track of what messages it has seen and only forwards those which it hasn't seen before.

Each message is of the form "ID:DATA" where ID is some arbitrary message identifier and DATA is the payload. In order to reduce memory usage, the agent stores only the message ID.

Note that there is no mechanism to expire old message IDs from the list of seen messages. There also isn't any mechanism for assigning message IDs.

*2) The events in the simulation:* Many packets are broadcast in the channel. As in the program:

```
$ns at 0.2 "$a(1) send_message 200 1
    {first message} $MESSAGE_PORT"
$ns at 0.4 "$a([expr $num_nodes/2])
    send_message 600 2
    {some big message} $MESSAGE_PORT"
$ns at 0.7 "$a([expr $num_nodes-2])
    send_message 200 3
    {another one} $MESSAGE_PORT"
```

At 0.2/0.4/0.7 seconds, hundreds of packets are sent so flooding easily occurs.

3) *analysis of the trace file*: There is 8 nodes in all, so the more neighbours of a node sending packets, the easier flooding may happen. When flooding happens, packets are lost.

At 0.41s, 0.44s, 0.71s and 0.72s, packets were lost. Among all the packets(600 packets) 15 packets were lost.

If we change the bandwidth to 0.1Mbps, over 20% of the packets were lost(30 in 120 packets).

If we choose 802.11(with DSDV) instead of simple MAC(with DumbAgent), only 6 packets in 650 were lost.

Moreover, if we choose 802.11(with AODV) instead of simple MAC(with DumbAgent), only 3 packets in 600 were lost.

Is AODV the best choice for the wireless networks? It will be solved in the next step of our project.

## 2) Simulation on large-scale web traffic

In search for examples of large scale networks, we find an ns-2 large scale wired network simulation(large-scale-web-traffic.tcl). This simulation provides us with a concept of the time/memory consumption of large scale network simulation.

In this simulation, 420 clients and 40 servers make tcp connections with each other(400 sessions each) throughout 4,200 seconds simulation time. The traffic is approximately 200,000 TCP connections with heavy-tailed connection sizes.

The author of the program runs the simulation with 62 MB memory for 1 hrs on FreeBSD 3.0, Pentium II Xeon 450 MHz PC with 1GB physical memory.

On Cygwin in windows XP, Pentium D 1.92GHz(Core Duo)PC with 2GB physical memory, the simulation lasted for 70 minutes with 44 MB memory.(Cygwin is really inefficient...)

The size of the output file is 2.40 GB, for every session of every node is logged.

## II. WHAT WE WILL DO NEXT

- 1) Combine the large scale model with wireless nodes and protocols, as our final model.
- 2) Apply more routing protocols to the final model and compare the simulation result, to know which is the best protocol to choose.
- 3) Get the relationship between the memory/time consumption and the number of nodes.