Appendix A

Network Simulator 2

A.1 Introduction

NS (Version 2) is an open source network simulation tool. It is an object oriented, discrete event driven simulator written in C++ and Otcl. The primary use of NS is in network research to simulate various types of wired/wireless local and wide area networks; to implement network protocols such as TCP and UDP, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and many more.

Ns2 is written in C++ and Otcl to separate the control and data path implementations. The simulator supports a class hierarchy in C++ (the compiled hierarchy) and a corresponding hierarchy within the Otcl interpreter (interpreted hierarchy).

In ns2, C++ is used for detailed protocol implementation and in general for such cases where every packet of a flow has to be processed. Using C++, anyone can implement new queuing discipline. Otcl is suitable for configuration and setup. Otcl runs quite slowly, but it can be changed very quickly making the construction of simulations easier. In ns2, the compiled C++ objects can be made available to the Otcl interpreter. The ready-made C++ objects can be controlled from the OTcl level.

Some of the important classes are:
The Class Tcl contains the methods that C++ code will use to access the interpreter. The class Tcl encapsulates the actual instance of the OTcl interpreter, and provides the methods to access and communicate with that interpreter.
The class provides methods for the following operations:

- obtain a reference to the Tcl instance;
- invoke OTcl procedures through the interpreter;
- retrieve, or pass back results to the interpreter;
- report error situations and exit in an uniform manner; and
- store and lookup "TclObjects".
- acquire direct access to the interpreter

Class TclObject is the base class for most of the other classes in the interpreted and compiled hierarchies. Every object in the class TclObject is created by the user from within the interpreter. An equivalent shadow object is created in the compiled hierarchy. The two objects are closely associated with each other.

The class TclClass defines the interpreted class hierarchy, and the methods to permit the user to instantiate TclObjects.

This compiled class (class TclClass) is a pure virtual class. Classes derived from this base class provide two functions: construct the interpreted class hierarchy to mirror the compiled class hierarchy; and provide methods to instantiate new TclObjects. Each such derived class is associated with a particular compiled class in the compiled class hierarchy, and can instantiate new objects in the associated class.

The class TclCommand is used to define simple global interpreter commands. This class provides just the mechanism for ns to export simple commands to the interpreter, that can then be executed within a global context by the interpreter. There are two functions defined in ~ns/misc.cc: ns-random and ns-version. These two functions are initialized by the function init_misc (void), defined in ~ns/misc.cc; init_misc is invoked by Tcl_ApplInit(void) during startup.

The class EmbeddedTcl contains the methods to load higher level built in commands that make configuring simulations easier. ns permits the development of functionality in either compiled code, or through interpreter code, that is evaluated at initialization. As example, the scripts ~tclcl/tcl-object.tcl or the scripts in ~ns/tcl/lib. Such loading and evaluation of scripts is done through objects in the class EmbeddedTcl.
The class Inst Var contains methods to access C++ member variables as OTcl instance variables.

This class defines the methods and mechanisms to bind a C++ member variable in the compiled shadow object to a specified OTcl instance variable in the equivalent interpreted object. The binding is set up such that the value of the variable can be set or accessed either from within the interpreter, or from within the compiled code at all times.

A.2 NS 2 for Wireless Network

NS 2 can be used to simulate wireless ad hoc network. A mobile node consists of network components like Link Layer (LL), Interface Queue (IfQ), MAC layer, the wireless channel nodes transmit and receive signals from etc.

*nam* is used to visualize the following output of a wireless network script,

- Mobile node position
- Mobile node moving direction and speed
- Control the speed of playback

A Simple wireless example in NS 2 is given below,

```bash
#Define Global Variables
set ns_ [new Simulator]
# Create a topology
set topo [new Topography]
# Define area 670x670
$topo load_flatgrid 670 670
#Define standard ns/nam trace
set tracefd [open 694demo.tr w]
$ns_trace-all $tracefd
set namtrace [open 694demo.nam w]
$ns_namtrace-all-wireless $namtrace 670 670
#Create “God”
```
#God is used to store an array of the shortest number of hops required to reach from one node to another.

    set god_ [create-god 3]

#Define how a mobile node should be created

    $ns_node-config -adhocRouting DSDV
        -llType LL
        -macType Mac/802_11
        -ifqLen 50
        -ifqType Queue/DropTail/PriQueue
        -antType Antenna/OmniAntenna
        -propType Propagation/TwoRayGround
        -phyType Phy/WirelessPhy
        -channelType Channel/WirelessChannel
        -topoInstance $topo
        -agentTrace ON
        -routerTrace OFF
        -macTrace OFF

#Create a mobile node and attach it to the channel

    set node [$ns_node]

    # Disable random motion
    $node random-motion 0

#Use “for loop” to create 3 nodes

    for {set i < 0} {$i<3} {incr i} {
        set node_(i) [$ns_node]
    }

#Define node movement model

    source movement-scenario-files

#Define traffic model

    source traffic-scenario-files
# Define node initial position in nam
for \{set i 0 \} \{$i < 3 \} \{ incr i \} {
     $ns\_initial\_node\_position $node\_(i) 20
}

# Simulation stop time
$ns\_at 200.0 \"$ns\_nam-end-wireless 200.00\" 

# Start simulation
$ns\_at 200.00
$ns\_halt
$ns\_run

# Mobile Movement Generator
setdest -n <num of nodes> -p pausetime -s <maxspeed> -t <simtime> -x <maxx> -y <maxy>

# Random movement
$node start

Source: See ns-2/indep-utils/cmu-scen-gen/setdest/

# Generating traffic pattern files
# CBR traffic
ns cbrgen.tcl [-type cb(tcp] [-nn nodes] [-seed seed] [-mc connections] [-rate]

# TCP traffic
ns tcpgen.tcl [-nn nodes] [-seed seed]

A.3 Running A New Routing Protocol
A new routing protocol for ns-2 has to be coded in C/C++. The output for this file can be incorporated into the simulator by specifying the file name in the Makefile and building ns-2 again. If the routing protocol involves a different packet format than what is defined in packet.h, this must also be specified in the header file.

Post Analysis
Post analysis is one of the important steps to analyze. After the simulation, it gives the trace file which contains the packet dump from the simulation. The format of this trace file for ad hoc wireless networks is as follows:

- **N**: Node Property
- **I**: IP Level Packet Information
- **H**: Next Hop Information
- **M**: MAC Level Packet Information
- **P**: Packet Specific Information

<table>
<thead>
<tr>
<th>Event</th>
<th>Abbreviation</th>
<th>Flag</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s: Send</td>
<td>-t</td>
<td>double</td>
<td>Time (* For Global Setting)</td>
</tr>
<tr>
<td></td>
<td>r: Receive</td>
<td></td>
<td>int</td>
<td>Node ID</td>
</tr>
<tr>
<td></td>
<td>d: Drop</td>
<td>-Ni</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f: Forward</td>
<td>-Nx</td>
<td>double</td>
<td>Node X Coordinate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Ny</td>
<td>double</td>
<td>Node Y Coordinate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Nz</td>
<td>double</td>
<td>Node Z Coordinate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Ne</td>
<td>double</td>
<td>Node Energy Level</td>
</tr>
<tr>
<td>Wireless</td>
<td></td>
<td>-Ni</td>
<td>string</td>
<td>Network trace Level (AGT, RTR, MAC, etc.)</td>
</tr>
<tr>
<td>Event</td>
<td></td>
<td>-Nw</td>
<td>string</td>
<td>Drop Reason</td>
</tr>
<tr>
<td></td>
<td>s: Send</td>
<td>-Hs</td>
<td>int</td>
<td>Hop source node ID</td>
</tr>
<tr>
<td></td>
<td>r: Receive</td>
<td>-Hd</td>
<td>int</td>
<td>Hop destination Node ID, -1, -2</td>
</tr>
<tr>
<td></td>
<td>d: Drop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>f: Forward</td>
<td>-Ma</td>
<td>hexadecimal</td>
<td>Duration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Ms</td>
<td>hexadecimal</td>
<td>Source Ethernet Address</td>
</tr>
</tbody>
</table>
Depending on the packet type, the trace may log additional information. The trace file is used to analyze different network performance parameters such as packet delivery ratio, throughput, end-to-end delay, round trip time etc.
Appendix B

Decision tree algorithm – ID5R

B.1 Introduction

Intrusion detection technology is found to be more effective once an efficient decision tree algorithm can be applied to the system, as decision tree can be converted into respective rules and can be utilized under knowledge base for intrusion detection. Decision tree is a fast classification method with top down divide and rule, starting from root, ability to learn classification intelligently is another behavior of decision tree. In case of decision tree, selecting test attributes and then to divide the sample set is very crucial. If the size of the sample set is very large then obviously branches and layers of the generated tree will also be large, as a result some unnecessary branches as well as abnormal result may generate, so these sort of branches should be pruned. Of course if there are several number of categories, then accuracy of decision tree is reduced.

For a set of data, which is increasing significantly, where non incremental decision tree algorithm cannot be applied, some incremental decision tree algorithm such as ID5R should be applied. Incremental algorithm is capable of building the concept step wise in a systematic manner as and when new training instances are available.

B.1.1 Introduction to ID5R

ID5R is an extension of ID3 algorithm, which maintains sufficient information to compute E-score for attribute at a node, based on lowest E-score, one of the attribute is selected as root node. Since it is an incremental decision tree algorithm, so instead
of discarding the sub tree every time after receiving new instance to the system it adopts the following:

- Re-structure the tree instead of rebuilding the tree.
- Pull up instead of simply deleting sub trees

Likewise, Leaf nodes contain a set of instance descriptions, Non-leaf nodes (decision nodes) contain set of potential candidate tests, each with positive and negative counts for each possible outcome and branches contain positive and negative counts for this outcome.

**B.1.2 Algorithm of ID5R**

- Start with empty tree $n$

1. If $n$ is empty:
   - create leaf
   - set leaf class to class of instance
   - set of instances: singleton set consisting of instance

2. If $n$ is a leaf and instance is of the same class:
   - Add instance to the set of instances

3. Otherwise:
   a) If $n$ is a leaf: expand it one level (choose test arbitrary)
   b) for all candidate tests at node $n$: update counts
   c) if the current test is not the best one among all candidate tests:
      i. Restructure tree $n$ so that the best test is now at root (Pull-Up)
      ii. Recursively reestablish best test in each sub tree (ignore path of instance)
   d) Recursively update tree $t$ along the path instance goes. Grow the branch if necessary.

To restructure the tree, the pull up algorithm is as follows

**Pull-Up a test attribute $A$**

1. If the test $A$ is already at the root, do nothing
2. Otherwise:
(a) Recursively pulls up A to the root of each immediate sub tree
(If necessary, expand leaves)
(b) Transpose the tree so that A is now in the root and the old root attribute is now
in the roots of all immediate sub trees.

It uses the following to measure the information gain:

\[
\text{Gain}(S,A) = \text{Entropy}(S) - \sum_{v \in \text{values } (A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)
\]

\[
\text{Entropy}(S) = \frac{|S|}{2} \log_2 \frac{|S_+|}{|S|} - \frac{|S_+|}{|S|} \log_2 \frac{|S_+|}{|S|} - \frac{|S_-|}{|S|} \log_2 \frac{|S_-|}{|S|}
\]

S: trainings set, A: test
S+ (S-): set of all positive (negative) examples in S
S_v : subset of all examples in S for which test A has value v
Appendix C

Protocol stack for TAODV and SAODV

Protocol format for SAODV:

**RREQ (Single) Signature Extension**

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Type | Length | Hash Function | Max Hop Count |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Top Hash |

...+-+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+

| Sign Method | H | Reserved | Padding Length |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Public Key |

...+-+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+

| Padding (optional) |

...+-+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+

| Signature |

...+-+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+

| Hash |

...+-+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+...+-+

**RREP (Single) Signature Extension**

```
### RREQ Double Signature Extension

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
```

- **Type**: 
- **Length**: 
- **Hash Function**: 
- **Max Hop Count**: 
- **Reserved**: 
- **Prefix Size**: 
- **Top Hash**: 
- **Sign Method**: 
- **Reserved**: 
- **Padding Length**: 
- **Public Key**: 
- **Padding (optional)**: 
- **Signature for RREP**: 
- **Signature**: 
- **Hash**: 
- **Signature for RREP**: 

### RREP Double Signature Extension

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
```

- **Type**: 
- **Length**: 
- **Hash Function**: 
- **Max Hop Count**: 
- **Reserved**: 
- **Prefix Size**: 
- **Top Hash**: 
- **Sign Method**: 
- **Reserved**: 
- **Padding Length**: 
- **Public Key**: 
- **Padding (optional)**: 
- **Signature for RREP**: 
- **Signature**: 
- **Hash**: 
- **Signature for RREP**:
RERR Signature Extension

| 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 |
| +--------------------------------------------|
| Type | Length | Reserved |
| +--------------------------------------------|
| Sign Method | Reserved | Padd Length |
| +--------------------------------------------|
| Public Key |
| +--------------------------------------------|
| Padding (optional) |
| +--------------------------------------------|
| Signature |
| +--------------------------------------------|
| Old Lifetime |
| +--------------------------------------------|
| 'Old Originator IP address |
| +--------------------------------------------|
| Sign Method 2 | Reserved | Padd Length 2 |
| +--------------------------------------------|
| Public Key 2 |
| +--------------------------------------------|
| Padding 2 (optional) |
| +--------------------------------------------|
| Signature of the new Lifetime and Originator IP address |
| +--------------------------------------------|
| Hash |
| +--------------------------------------------|

168
**RREP-ACK Signature Extension**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

... | Type | Length |

| Sign Method | Reserved | Public Key |

... | Padding (optional) |

... | Signature |

---

**Protocol format for TAODV**

**Trust Request (TREQ) Message Format**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

... | Type | #of Recommendee | Reserved |

... | Recommendee IP Address 1 |

... | Recommendee IP Address n |

---

**Trust Reply (TREP) Message Format**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

... | Type | #of Recommendee | Reserved |

... | Recommendee IP Address 1 |

... | Recommendee IP Address n |

---
<table>
<thead>
<tr>
<th>Recommendee IP Address n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opinion about Recommendee 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Opinion about Recommendee n</td>
</tr>
</tbody>
</table>