

# Design Routing Protocol Performance Comparison in NS2: AODV comparing to DSR as Example

Yinfei Pan

*Department of Computer Science  
SUNY Binghamton  
Vestal Parkway East, Vestal, NY 13850*

## Abstract

*There are already many projects try to tell us how to use NS-2 easily, while not as that difficult as its official manual says. However, how to easily use NS-2 to do performance evaluation is still lacking. Usually, a trace file which comes out from one time NS-2 simulation is larger than 600MB, to analysis this huge file definitely will cost great time. Till now, there are many evaluation methods, but use them is very time cost and may not exactly what the simple version we want. This paper is to let people who use NS-2 can easily do the work of network post simulation. Thus make it easier and faster in making NS-2 more practical. Furthermore, this paper apply this performance evaluation method on the comparison of two on-demand source routing protocols AODV and DSR, and shows DSR is somewhat suitable for sensor work applications*

## 1. Introduction

NS-2 is an object-oriented simulator developed as part of the VINT project at the University of California in Berkeley. The project is funded by DARPA in collaboration with XEROX Palo Alto Research Center (PARC) and Lawrence Berkeley National Laboratory (LBNL). NS-2 is extensively used by the networking research community. It provides substantial support for simulation of TCP, routing, multicast protocols over wired and wireless (local and satellite) networks, etc. The simulator is event-driven and runs in a non-realtime fashion. It consists of C++ core methods and uses Tcl and Object Tcl shell as interface allowing the input file (simulation script) to describe the model to simulate. Users can define arbitrary network topologies composed of nodes, routers, links and shared media. A rich set of protocol objects can then be attached to nodes, usually as agents. It had already become the "de facto" standard in networking research.[1, 2]

Although NS is fairly to use once you get to know the simulator, it is quite difficult for a first time user, because there are few user-friendly manuals. Even though there is lots documentation written by the developers who has in depth explanation of simulator, it is written with the depth of a skilled NS user.

There already are many user friendly tutorials to make how to use ns-2 easily [3, 4]. They give a new user some basic idea of how the simulator works, how to setup simulation networks, where to look for further information about network components in simulator codes, how to create new network components, and so on. They are mainly focus on simple examples and brief explanations. Although all the usage of the simulator or possible network simulation setups may not be covered in those projects, the projects also help a new user to get quick started. However, with this good tutorials material available today, there still lack documents to ease the analysis work of post-simulation. Since ns-2 is an academic project, the main purpose is for evaluating the existing network's performance or the performance of network with new design of component [6, 7]. Thus, to make progress on this problem, this paper is to let people who use ns-2 can easily do the work of network post simulation analysis, which will further make ns-2 more practical on doing scalable simulation.

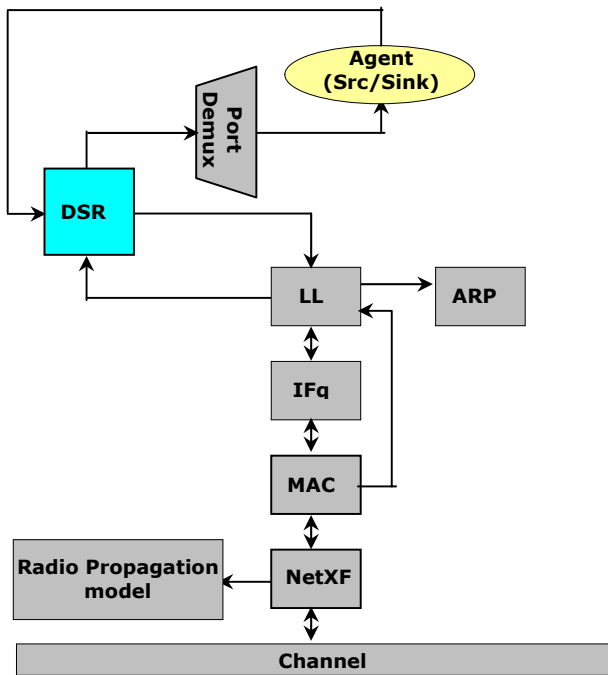
In the rest of this paper, section 2 will discuss how to conduct a wireless simulation in ns-2; section 3 will describe about the trace file format of ns-2, and then section 4 will do analysis on two sample routing protocols, AODV and DSR, and also show how our performance evaluation mode can work on the analysis of them.

## 2. Wireless simulation in NS-2

### 2.1. Software structure and mechanism of NS-2

The key to get to know ns-2 is it is a discrete event network simulator. In ns-2 network physical activities are translated to events, events are queued and processed in the order of their scheduled occurrences. And the simulation time progresses with the events processed. And also the simulation “time” may not be the real life time as we “inputted”.

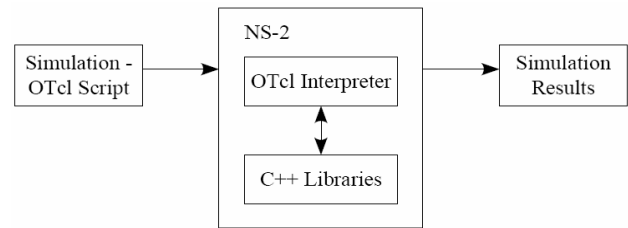
But, why is ns-2 that useful, what kind of work can be done by ns-2, it can model essential network components, traffic models and applications. Typically, it can configure transport layer protocols, routing protocols, interface queues, and also link layer mechanisms. We can easily see that this software tool in fact could provide us a whole view of the network construction, meanwhile, it also maintain the flexibility for us to decide. Thus, just this one software can help us simulate nearly all parts of the network [1-5]. This definitely will save us great amount of cost invested on net work constructing. The following Figure 1 shows a layered structure which ns-2 can simulate for us.



**Figure 1 ns-2 simulate layered structure of network**

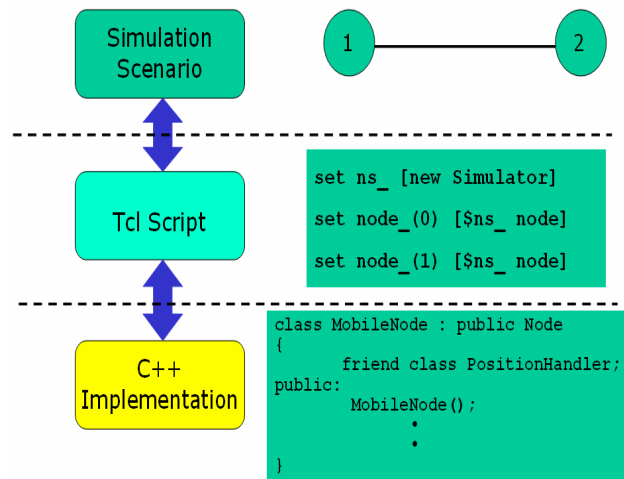
After the simulation finish, the way ns-2 used to present the most details information on that much network layer is that it provides us a huge trace file recording all the events line by line in it. So, now we see why event driven mechanism is used in ns-2, since it really could maintain the things ever happened as records. And we can trace these records to evaluate the performance of special stuffs in our network, such as

routing protocol, Mac layer load, and so on.



**Figure 2 data flow for one time simulation**

As Figure 2 shows, for the data flow of one time simulation in ns-2, the user input an OTcl source file, the OTcl script do the work of initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler. And then, this OTcl script file will be passed to ns-2, in this view, we can treat ns-2 as Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup module libraries. And then the detail network construction and traffic simulation will be actually done in ns-2. After a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, and the data can be used for simulation analysis [3, 5].



**Figure 3 Layered structure from the ns-2 developer view**

From the NS-2 developer view, Figure 3 shows the layered architecture of ns. The event schedulers and most of the network components are implemented in C++ and available to Tcl Script, thus the lowest level of NS-2 is implemented by C++, and the Tcl script level is

on top of it to make simulation stuffs much easier to be conducted. Then, upon the Tcl level, we see the overview of the network. That is the simulation scenario. These all things combined as so called ns-2 software.

## 2.2. Parts needed by one simulation in NS-2

To successfully carry out one simulation, we must first tell ns-2 things it may need from us for one simulation. So what we need is the follow three necessary items:

- 1) Appearance of the network: the whole topology view of sensor network or mobile network, this includes the position of nodes with (x, y, z) coordinate, the node movement parameters, the movement starting time, the movement is to what direction, and the node movement speed with pausing time between two supposed movement.
- 2) Internal of the network: Since the simulation is on the network traffic, so it is important we tel the ns2 about which nodes are the sources, how about the connections, what kind of connection we want to use.
- 3) Configuration of the layered structure of each node in the network, this includes the detail configuration of network components on sensor node, and also we need to drive the simulation, so we need to give out where to give out the simulation results which is the trace file, and how to organize a simulation process.

## 2.3. Writing tcl to run simple wireless simulations

In this section, we then will present step by step of how to do all the things as needed by one simulation in ns-2 with one tcl scripts sequence [1-5]:

Step 1. Create an instance of the simulator:

```
set ns_ [new Simulator]
```

Step.2. Setup trace support by opening file "trace\_bbtr.tr" and call the procedure trace-all

```
set tracefd [open trace_bbtr.tr w]
$ns_ trace-all $tracefd
```

Step 3. Create a topology object that keeps track # of all the nodes within boundary

```
set topo [new Topography]
```

Step 4. The topography is broken up into grids and the default value of grid resolution is 1. A different value can be passed as a third parameter to load\_flatgrid {}.

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

Step 5. Create the object God, "God (General Operations Director) is the object that is used to store global information about the state of the environment, network or nodes. The procedure create-god is defined in \$NS2\_HOME/tcl/mobility/com.tcl, which allows only a single global instance of the God object to be created during a simulation. God object is called internally by MAC objects in nodes, so we must create god in every cases.

```
set god_ [create-god $val(nn)]
```

Step 6. Before we can create node, we first needs to configure them. Node configuration API may consist of defining the type of addressing (flat/hierarchical etc), for example, the type of adhoc routing protocol, Link Layer, MAC layer, IfQ etc.

```
$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) \
    -channe
    -channel [new $val(chan)] \
    -topoInstance $topo \
    -agentTrace ON \
    -routerTrace ON \
    -macTrace OFF \
    -movementTrace OFF
```

Step 7. Create nodes and the random-motion for nodes is disabled here, as we are going to provide node position and movement (speed & direction) directives next

```
for {set i 0} {$i < $val(nn)} {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 0
    # Disable random motion
}
```

Step 8. Give nodes positions to start with, Provide initial (X,Y, for now Z=0) co-ordinates for node\_(0) and node\_(1). Node0 has a starting position of (5,2) while Node1 starts off at location (390,385).

```
$node_(0) set X_ 5.0
```

```

$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 390.0
$node_(1) set Y_ 385.0
$node_(1) set Z_ 0.0

```

Step 9. Setup node movement as the following example, at time 50.0s, node 1 starts to move towards the destination (x=25, y=20) at a speed of 15m/s. This API is used to change direction and speed of movement of nodes.

```
$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0"
```

Step 10. Setup traffic flow between the two nodes as follows: TCP connections between node\_(0) and node\_(1)

```

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp
$ns_ attach-agent $node_(1) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 10.0 "$ftp start"

```

Step 11. Define stop time when the simulation ends and tell nodes to reset which actually resets their internal network components. In the following case, at time 150.0s, the simulation shall stop. The nodes are reset at that time and the "\$ns\_ halt" is called at 150.0002s, a little later after resetting the nodes. The procedure stop{} is called to flush out traces and close the trace file.

```

for {set i 0} {$i < $val(nn)} {incr i} {
    $ns_ at 150.0 "$node_($i) reset";
}

$ns_ at 150.0001 "stop"
$ns_ at 150.0002 "puts \"NS EXITING...\"; $ns_ halt"

proc stop {} {
    global ns_ tracefd nf
    $ns_ flush-trace
    close $tracefd
    close $nf
}

```

Step 12. Finally the command to start the simulation

```
puts "Starting Simulation...\n" $ns_ run
```

So, these 12 steps could finish one time simulation,

and we can pack these 12 steps into one tcl file and do the simulation. However, there exist some problems on such kind of use on typical network performance test situations. Performance testing usually needs to be scalable in the number of nodes and network transmitting packets. Suppose for one network there are hundreds of nodes, we need to set all of the nodes' positions and their movement, this a huge amount of workload, also, suppose we need to setup all the possible sources and destinations and even connections, also is a huge workload, Furthermore, even if we can set them, we cannot guarantee our input is randomly select, which is necessary for a fair comparison.

## 2.4. Network scenario and traffic generating

As the problem may exist in above, we then will need nodes placement and network traffic can be generated automatically and thus could meet the demand of scalable performance test for a specific network configuration. There are some third part tools that could help us to do so.

### 2.4.1. Network scenario generating

For nodes positions and their movement, we can generate a file with the statements which set nodes' positions and nodes movement using CMU generator. It is under

\$NS2\_HOME/indep-utils/cmu-scen-gen/setdest. But, before we use it, we need first run "make" to create executable "setdest" program.

In fact, this is a third party which is CMU's version auxiliary scenario creation tool. it used system dependent /dev/random and made calls to library functions initstate() for generating random numbers. The usage of this executable command is:

```

/setdest [-n num_of_nodes] [-p pausetime] [-s maxspeed]
[-t simtime] [-x maxx] [-y maxy] > [scenario_output_file]

```

For example, if the command we use is like:

```
./setdest -n 20 -p 2.0 -s 10.0 -t 200 -x 500 -y 500 > scen-20-test
```

This means, the topology boundary is 500m X 500m, the scenario has 20 nodes with nodes' max moving speed of 10.0m/s and the pause between movements is 2s. Also, Simulation will stop in 200s, and finally, output the generate tcl statements into file whose name is scen-20-test. Some fragments of scen-20-test are shown in Figure 4 below

```

$node_(0) set X_ 31.037289988193 $god_ set-dist 0 20 3
$node_(0) set Y_ 493.363103158662 $god_ set-dist 0 21 2
$node_(0) set Z_ 0.000000000000 $god_ set-dist 0 22 3
$node_(1) set X_ 254.222799064206 $god_ set-dist 0 23 2
$node_(1) set Y_ 486.833590260098 $god_ set-dist 0 24 2
$node_(1) set Z_ 0.000000000000 $god_ set-dist 0 25 3
$node_(2) set X_ 217.323676613934 $god_ set-dist 0 26 2
$node_(2) set Y_ 296.524595510474 $god_ set-dist 0 27 3
$node_(2) set Z_ 0.000000000000 $god_ set-dist 0 28 3
$node_(3) set X_ 480.021754010703 $god_ set-dist 0 29 2
$node_(3) set Y_ 242.442866770578 $god_ set-dist 0 30 3
$node_(3) set Z_ 0.000000000000 $god_ set-dist 0 31 2
$node_(4) set X_ 108.023254426391 $god_ set-dist 0 32 1
$node_(4) set Y_ 254.517771755275 $god_ set-dist 0 32 1
$node_(4) set Z_ 0.000000000000 $god_ set-dist 0 32 1

```

**Figure 4 fragments of scenario generating result**

From figure 4, we notice that the node movement is the same pattern as we described before. That is because this scenario generating program uses GOD. Directives for GOD are present as well in node-movement file. The General Operations Director (GOD) object is used to store global information about the state of the environment, network, or nodes that an omniscient observer would have, but that should not be made known to any participant in the simulation. Currently, the god object is used only to store an array of the shortest number of hops required to reach from one node to another. The god object does not calculate this on the fly during simulation runs, since it can be quite time consuming. The information is loaded into the god object from the movement pattern file. And the setdest program generates node-movement files using the random waypoint algorithm. These files already include the lines to load the god object with the appropriate information at the appropriate time.

#### 2.4.2. Network traffic generating

For network traffic generating, what are generated are also statements on such as sources, connections, and so on. This work could be done as a tcl file, which is in \$SNS2\_HOME/indep-utils/cmuc-scen-gen/cbrgen.tcl. Since this can be easily read and modified, if you want, you can just simply modify it so that the scenario be generated will be more suitable to what your need. For this network traffic generating tool, random traffic connections of TCP and CBR can be setup between nodes. It can be used to create CBR and TCP traffics connections between wireless nodes. In order to create a traffic-connection file, we need to define the type of traffic connection (CBR or TCP), the number of nodes and maximum number of connections to be setup

between them, a random seed and incase of CBR connections, a rate whose inverse value is used to compute the interval time between the CBR packets. So the command line is:

```

ns cbrgen.tcl [-type cbr/tcp] [-nn nodes] [-seed seed]
              [-mc connections] [-rate rate]

```

Here, “-type cbr/tcp” means define the type of traffic connection, “-nn nodes” means the number of nodes could be used, “-mc connections” means maximum number of connections to be setup between those nodes, “-seed seed” means a random seed, if it not equal to 0, the traffic pattern will reappear if all the other parameters are the same. “-rate rate” means a rate whose inverse value is used to compute the interval time, which easily to say is packets sending rate.

For an example:

```

ns cbrgen.tcl -type cbr -nn 20 -seed 1.0 -mc 20 -rate 4.0
              > cbr-20-test

```

means create a CBR connection pattern between 20 nodes, having maximum of 20 connections, with a seed value of 1.0 and a rate of 4.0 pkts/second.

Figure 5 shows one fragment of traffic generating output

```

# 1 connecting to 2 at time 2.5568388786897245
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
#
# 4 connecting to 5 at time 56.333118917575632
#
set udp_(1) [new Agent/UDP]
$ns_ attach-agent $node_(4) $udp_(1)
set null_(1) [new Agent/Null]
$ns_ attach-agent $node_(5) $null_(1)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 512
$cbr_(1) set interval_ 0.25
$cbr_(1) set random_ 1
$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(1)
$ns_ at 56.333118917575632 "$cbr_(1) start"

```

**Figure 5 Traffic generating output**

### 2.4.3. add generating results to tcl file

So, we now get two scenario files, they are all formed by tcl scripts, so our purpose now is to add these results into the main tcl file, so that they can get run and the scalable scenario would be achieved.

So, add them to main tcl file has the following steps:

1. Add two variables in parameter options in the main tcl file

```
set val(sc) "/home/ypan3/sensortcl/scen-20-test "  
set val(cp) "/home/ypan3/sensortcl/cbr-20-test"
```

Here, "/home/ypan3/sensortcl/" could be on directory you store all your tcl files

2. In the main tcl file, just replace all the tcl statements which are related to node placement and movement, and also traffic creating with the following two statements

```
source $val(sc)  
source $val(cp)
```

Now, after adding the generating results to tcl file, we finished one time simulation, but we need to do the test multiple time with different control parameters we want. So the next we will discuss how we implement automatic test.

### 2.5. Do generating automatically for performance test

To enable multiple tests with different control parameters, the best way is use shell script to run different commands as we want [8-11]. It will be very clear with the following shell script which generates different scenarios with different number of nodes.

```
for sim_time in 1 2 3  
do  
    mkdir Scenario_${sim_time}  
    for node_num in 50 100 150  
    do  
        ./setdest -v 1 -n ${node_num} -p 400 -M 5 -t  
        200 -x 500 -y 500 >
```

```
Scenario_${sim_time}/scen_${node_num}_4  
00_5_200_500_500
```

done

done

Then, the same way could be applied to generate different traffic patterns.

### 2.6. Accumulate extracted results

For a more precise experiment result, we usually try to run the same test for several times, and get the average. So, the following pseudo code of shell script will automatically make several times run of the tests.

```
for sim_time in 1 2 3  
do  
    for node in 50 100 150  
    do  
        #go to the specific directory with specific number of node  
        cd ${tracefile_directory}  
        rm -f trace_bbtr.tr # this is the trace file  
        ns main.tcl  
        rm -f result.txt  
        perl cal_trace.pl  
        # do trace file analysis in a perl script file  
        cat result.txt >> result1.txt  
        # append the results  
    done  
done
```

Please notice that we run a perl script each time on the trace file. It is in fact to do the trace file analysis. We will discuss this issue later in this paper.

## 3. Trace file analysis

There are some terms on the performance matrix we defined in this paper, so the analysis will be centered on these terms.

First, Packet Delivery Rate, this equals to "Total packets successfully received / Total packets sent"; Second, Average Delay, this means "Sum (for each i equal to packet number, (packet i received - time packet i sent time)) / Total packets transmitted"; Third, Average Routing Load, this means "Total routing control packets / Simulation time".

```

r 2.556838879 _1_ RTR --- 0 cbr 512 [0 0 0 0] ----- [1:0 2:0 32 0] [0] 0 1
s 2.556838879 _1_ RTR --- 0 AODV 48 [0 0 0 0] ----- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]] (REQUEST)
r 2.557779023 _36_ RTR --- 0 AODV 48 [0 ffffffff 1 800] ----- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]] (REQUEST)
r 2.557779058 _37_ RTR --- 0 AODV 48 [0 ffffffff 1 800] ----- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]] (REQUEST)
r 2.557779080 _21_ RTR --- 0 AODV 48 [0 ffffffff 1 800] ----- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]] (REQUEST)
r 2.557779135 _41_ RTR --- 0 AODV 48 [0 ffffffff 1 800] ----- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]] (REQUEST)
r 2.557779214 _45_ RTR --- 0 AODV 48 [0 ffffffff 1 800] ----- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]] (REQUEST)
r 2.557779280 _29_ RTR --- 0 AODV 48 [0 ffffffff 1 800] ----- [1:255 -1:255 30 0] [0x2 1 1 [2 0] [1 4]] (REQUEST)

```

(a) Trace file when routing with AODV

```

s 7.707201759 _1_ RTR --- 26 cbr 532 [0 0 0 0] ----- [1:0 2:0 32 2] [22] 0 1
s 7.710612960 _7_ RTR --- 25 DSR 32 [0 0 0 0] ----- [7:255 9:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 7.712911244 _2_ RTR --- 26 cbr 532 [13a 2 1 800] ----- [1:0 2:0 32 2] [22] 1 1
r 7.712911244 _2_ AGT --- 26 cbr 512 [13a 2 1 800] ----- [1:0 2:0 32 2] [22] 1 1
r 7.714547766 _33_ RTR --- 25 DSR 32 [0 ffffffff 7 800] ----- [7:255 9:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 7.714547944 _11_ RTR --- 25 DSR 32 [0 ffffffff 7 800] ----- [7:255 9:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 7.714548014 _36_ RTR --- 25 DSR 32 [0 ffffffff 7 800] ----- [7:255 9:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]
r 7.714548052 _44_ RTR --- 25 DSR 32 [0 ffffffff 7 800] ----- [7:255 9:255 32 0] 1 [1 1] [0 1 0 0->0] [0 0 0 0->0]

```

(b) Trace file when routing with DSR

Figure 6 Snapshot of the trace file

### 3.1. Trace file format [12]

Figure 6 shows the trace files for routing with AODV protocol and routing with DSR separately.

Here, we need to point out that for the first column, r means “receive”, s means “send”, d means “drop”, and f means “forward”. And the second column is the time when event was triggered. The third column is the node

number, the fourth column is this event is in which level, “RTR” means in routing level, “AGT” means in agent level, “MAC” means in MAC level. Then, the seventh column is the package type, the eighth column is the event ID.

Details on the trace file format description will be show as the following figures.

For common header, we show it in figure 7

Event	Abbreviation	Type	Value
Wireless Event	s: Send r: Receive d: Drop f: Forward	%.9f %d (%6.2f %6.2f) %3s %4s %d %s %d [%x %x %x %x]	%.9f %d %3s %4s %d %s %d [%x %x %x %x]
		double	Time
		int	Node ID
		double	X Coordinate (If Logging Position)
		double	Y Coordinate (If Logging Position)
		string	Trace Name
		string	Reason
		int	Event Identifier
		string	Packet Type
		int	Packet Size
		hexadecimal	Time To Send Data
		hexadecimal	Destination MAC Address
		hexadecimal	Source MAC Address
		hexadecimal	Type (ARP, IP)

Figure 7 the common header of trace file

For IP trace header, it is shown in Figure 8

IP Trace	----- [Nd:Nd:Nd:Nd:Nd] (REQUEST)	
	int	Source IP Address
	int	Source Port Number
	int	Destination IP Address
	int	Destination Port Number
	int	TTL Value
	int	Next Hop Address, If Any

**Figure 8 IP trace header**

For AODV protocol specific trace, we show is in Figure 9

AODV Trace	[0x%x %d %d %d %d] [Nd Nd] (REQUEST)	
	hexadecimal	Type
	int	Hop Count
	int	Broadcast ID
	int	Destination
	int	Destination Sequence Number
	int	Source
	int	Source Sequence Number
	[0x%x %d %d %d %d] [%f] (%s)	
	hexadecimal	Type
	int	Hop Count
	int	Destination
	int	Destination Sequence Number
	double	Lifetime
	string	Operation (REPLY, ERROR, HELLO)

**Figure 9 AODV trace**

DSR Trace	%d [Nd Nd] [Nd Nd Nd Nd->Nd] [Nd Nd Nd Nd->Nd]	
	int	Number Of Nodes Traversed
	int	Routing Request Flag
	int	Route Request Sequence Number
	int	Routing Reply Flag
	int	Route Request Sequence Number
	int	Reply Length
	int	Source Of Source Routing
	int	Destination Of Source Routing
	int	Error Report Flag (?)
	int	Number Of Errors
	int	Report To Whom
	int	Link Error From
	int	Link Error To

**Figure 10 DSR trace**

### 3.2. Analysis trace file with perl script

Since perl script is very efficient in match data file which has the form of line records [13, 14]. And is very convenient in extract numeric variables from strings, also it is easy to do the computation as C programming language does. So, we use perl script to analysis the trace file and direct compute out the performance matrix we provide at the beginning of section 3.

The pseudo code of the perl script shows as follow:

```

open(FileHndl, "trace_bbtr.tr")
while($line = <FileHndl> )
{
    if($line =~
/(w)s(d*.d*)s|(d*)|_s*|w*s*(S*)s*(d*)s*(cbr)s|d*s|
/[d*|w*s|d*|w*s|d*|w*s|d*|w*s|_s*|S*s*|/(d*)|.d*s*(d*)/)
    {
        #print "$1, $2, $3, $4, $5, $6, $7. \n";
        if($1 eq "s")
        {
            $num_of_s ++;
        }
        if($1 eq "s" && $3 eq $6)
        # Current node is its destination node
        {
            $pkt_s_time[$4] = $2;
            # $4 is event id
            $num_of_t ++;
        }
    }
    ....
    ....
    if($line =~
/(w)s(d*.d*)s|(d*)|_s*|w*s*(S*)s*(d*)s*(AO
DV)/)
    {
        #print "$1, $2, $3, $4, \n";
        if ($flag eq 0)
        {
            $simulation_start_time = $2;
            start time of begin transmit pkts
            $flag = 1;
        }
        if ($1 eq "r" && $4 ne "END")
        {
            $pkt_rcv_time = $2;
            #update to newest pkts receive time
            $num_of_routing ++;
        }
    }
}

```



So, from the pseudocode, it is easily to see that the perl script is to get the trace file line by line. And the statement:

```
if($line =~
/(\w)\s(d*\.\d*)\s|_(d*)\s|_s*\w*\s*\s*\s*(d*)\s*(cbr)\s\d*\s|
|/d*\w*\s\d*\w*\s\d*\w*\s\d*\w*\s|/s*\s*\s*/(d*)\s\d*\s*(d*)/)
```

works as a filter when try to get a line from the trace file.

So, it can easily draw out the packets we want to count, such as CBR packets, AODV routing control packets, DSR routing control packets, and so on.

#### 4. Performance Comparison Example on AODV & DSR

The following will be on AODV and DSR comparison, the main part will be on the test results which were got from the above performance evaluation code.

##### 4.1. Brief on AODV and DSR

For AODV [15-19], it performs Route Discovery using control messages route request (RREQ) and route reply (RREP) whenever node wishes to send packet to destination. To control network wide broadcasts of RREQs, the source node use an expanding ring search technique. The forward path sets up in intermediate nodes in its route table with a lifetime association using RREP. When either destination or intermediate node moves, a route error (RERR) is sent to the affected source nodes. When source node receives the (RERR), it can reinitiate route discovery if the route is still needed. Neighborhood information is obtained from broadcast Hello packet. An important feature of AODV is the maintenance of timer-based states in each node, regarding utilization of individual routing table entries. A routing table entry is “expired” if not used recently. A set of predecessor nodes is maintained for each routing table entry, indicating the set of neighboring nodes that use that entry to route data packets. These nodes are notified with RERR packets when the next hop link breaks. Each predecessor node, in turn, forwards the RERR to its own set of predecessors, thus effectively erasing all routes using the broken link. In contrast to DSR, RERR packets in AODV are intended to inform all sources using a link when a failure occurs. Route error propagation in AODV can be visualized conceptually as a tree whose root is the node at the point of failure and all sources using the failed link as the leaves.

For DSR [20-22], this protocol has two mechanisms: Route Discovery and Route Maintenance. The source route is needed when some node originates a new packet destined for some node by searching its route cache or initiating route discovery using ROUTE REQUEST and ROUTE REPLY messages. On detecting link break, DSR sends ROUTE ERROR message to source for new route. DSR is a simple and efficient routing protocol designed for use in multi hop wireless ad hoc networks of mobile nodes. Network is self-organizing and self-configuring when using DSR. When the nodes in an ad hoc network move and join the network while forwarding packets, all routing is automatically determined and maintained by DSR. DSR allows nodes to dynamically discover a source route across multiple network hops to any destination. Each data packet sent then carries in its header the complete, ordered list of nodes through which the packet must pass, allowing packet routing to be trivially loop free and avoiding the need for up-to-date routing information in the intermediate nodes. So for DSR, the overhead incurred by performing a new Route Discovery can be avoided when the caching of multiple routes to a destination occurs.

##### 4.2. Comparison on AODV and DSR

The DSR and the AODV protocol are two dynamic routing protocols that initiate routing activities for ad hoc networks on an on demand basis [23, 24]. These protocols were designed for reducing the routing loading in networks. The routing mechanism in DSR uses source routing, while AODV uses a table driven routing framework and destination sequence numbers. AODV relies on certain timer-based activities while DSR does not rely on such options. In DSR the sender knows the hop-by-hop route to the destination because of the use of source routing in the protocol. In DSR, the routes are stored in route cache. The packet header contains the source route. Route Discovery is used to dynamically determine a route when the route is not known. Route Request packets are sent to flood the network and Route Error packets are sent when any link in the source route is broken. DSR makes use of source routing and route caching. DSR uses Route Maintenance mechanism to repair routes that get broken when sending packets from the sender to the destination. Packet Salvaging, Automatic Route Shortening, Increased Spreading of Route Error Messages are some of the mechanisms, which are used under Route Maintenance.

AODV discovers routes on an on-demand basis

using a similar route discovery process as in DSR. AODV uses traditional routing tables, one entry per destination for maintaining routing information. DSR on the other hand maintains multiple route cache entries for each destination. To propagate route reply back to the source and to route data packets to the destination, AODV relies on routing table entries. Sequence numbers at each destination determines freshness of routing information and prevents routing loops. Routing packets carry the sequence numbers. The maintenance of timer-based states in each node for utilization of individual route entries is an important feature of AODV protocol. Sets of predecessor nodes are maintained for each routing table entry, which indicates the set of neighboring nodes. Route Error packets are used to notify these nodes when the next-hop link breaks. All the routes using the broken link are erased when the route error packets are sending to its own set of predecessors. Route Error packets in AODV are intended to inform all sources using a link when a failure occurs. In AODV, Route Error propagation is visualized as a tree structure where the root is the node at the point of failure and all sources using the failed link as leaves. An optimizing technique in AODV to control Route Request flood in the route discovery process is to use an expanding ring search to discover routes to unknown destination. DSR with the influence of source routing and promiscuous listening of data packet transmissions has access to a significantly greater amount of routing information than AODV. AODV can gather only limited amount of routing information. DSR uses route caching aggressively by replying to all requests reaching a destination from a single request cycle. In AODV, the destination replies only once to the request arrive first. The rest of the requests are ignored. Since DSR does not have any mechanism for the expiration of stale routes stored in the cache, some of these stale routes may start polluting other caches. AODV when faced with the choice of stale routes would choose the fresher one. The entry in the routing table if not used recently gets expired [25-27].

In all, DSR allow cache more paths from a source to a destination, while AODV just use the path first discovered. Thus, DSR have significant greater amount of routing information than AODV. Meanwhile, DSR has access to many alternate routes which saves route discovery floods, the performance then will be better if they are actually in use. DSR doesn't contain any explicit mechanism to expire stale routes in the cache. Choosing stale routes under stressful situation is normal thing in DSR. So, AODV is more conservative

while DSR is more aggressive in using the past information. Which is better is hard to say [25-27].

### **4.3. The expect performance comparison results on AODV and DSR**

Accord to the discussion on AODV and DSR above, our performance evaluation mode should give out the expected result on the following matrix:

For routing load, DSR should be lower than AODV (in terms of number of packets) due to its aggressive caching (though more replies and errors) and another reason is that AODV is dominated by RREQ packets.

For packets delivery rate and delay, DSR should be better if less stressful traffic load. Since caching only provide significant benefits to a certain extent. Stale caches are chosen in high loads which will cause unnecessary bandwidth consumption and pollution of caches in other nodes under high load with DSR.

For average delays, which may determine by network congestion, bandwidth cost by routing overhead, and routing error, so it is a little hard to say but DSR may be better in most cases since the routing overhead is extremely great with AODV.

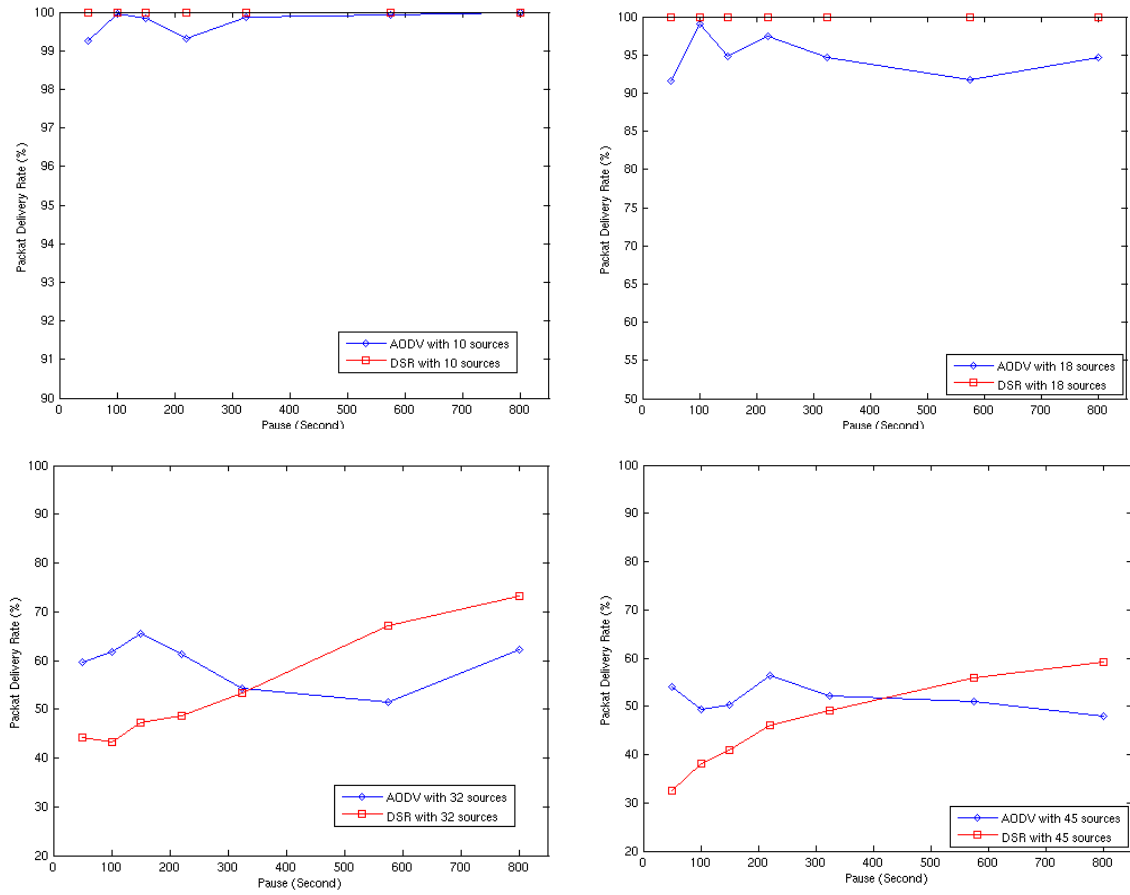
## **5. Apply our performance comparison method on AODV & DSR**

### **5.1. Simulation design**

The simulation is done under ns-2 using the method we discussed previously. The basic configuration is that our testing is in a 500 \* 500 square with total 50 nodes. The traffic sources are CBR (constant bit rate), 512-byte as data packets, sending rate is 4 pkts/second. To use CBR is for a fair comparison purpose, since bit rate vary will make the data packets traffic load unpredictable, which situation we do not want it happen. The number of sources we use are 10, 18, 32 and 45 separately. The node movement speed is set to from 0 to 5 which will be closer to the sensor network's application. The mobility are done with various pause time: 50, 100, 150, 220, 325, 575, 800 seconds (pretty high pause time, since we do not want too much mobility occurs in sensor networks application), and the MAC we employ is 802.11 MAC

### **5.2. Simulation result**

#### **5.2.1. Packet delivery rate**

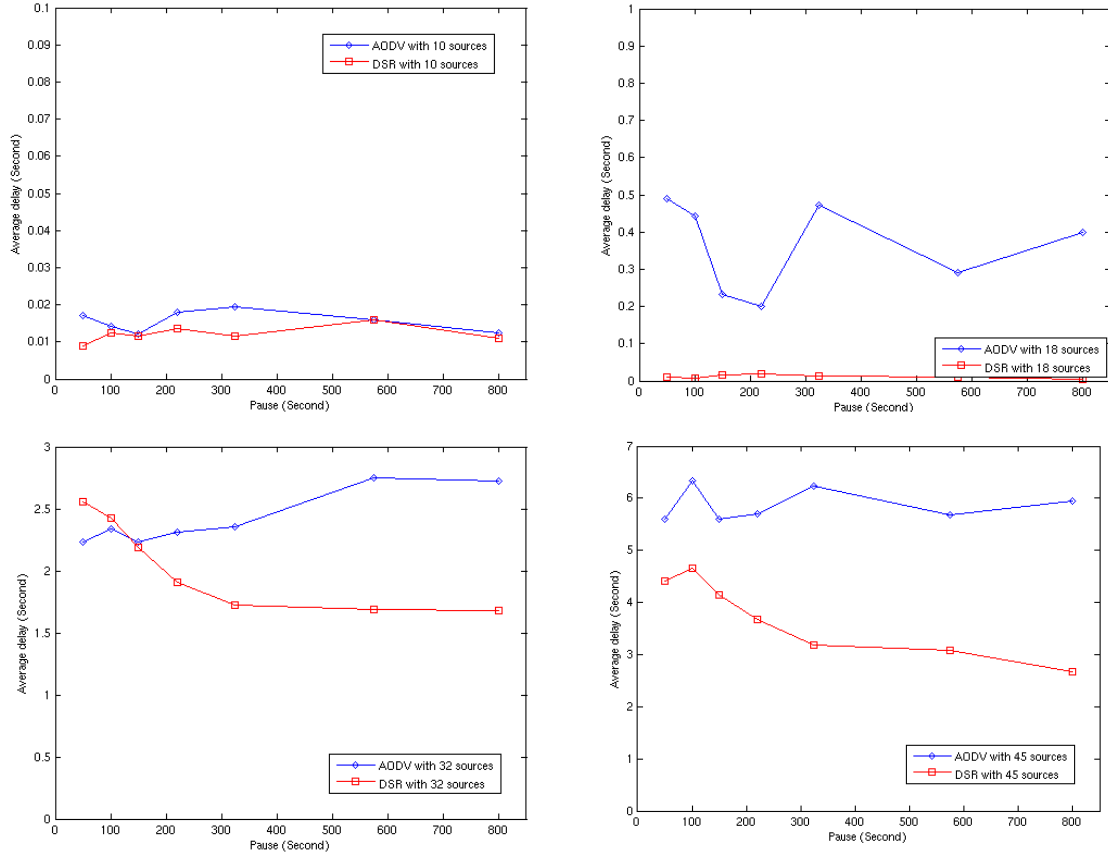


**Figure 11 Packet delivery rate comparison**

Figure 11 shows the experiment results of packets delivery rate. We can see, under low traffic load as 10 sources and 18 sources. DSR out perform the AODV, but when the sources become more, DSR with small pause time which means high mobility began to perform worse than AODV. This result is what we want to see, since AODV have more routing control packets but may always choose the fresh route, while DSR with

smaller number of routing packets under stressfully situation with the network topology continue to change from time to time, will be inclined to choose wrong routes, thus lower the packets delivery rate.

### 5.2.2. Average delay



**Figure 12 Average delay comparison**

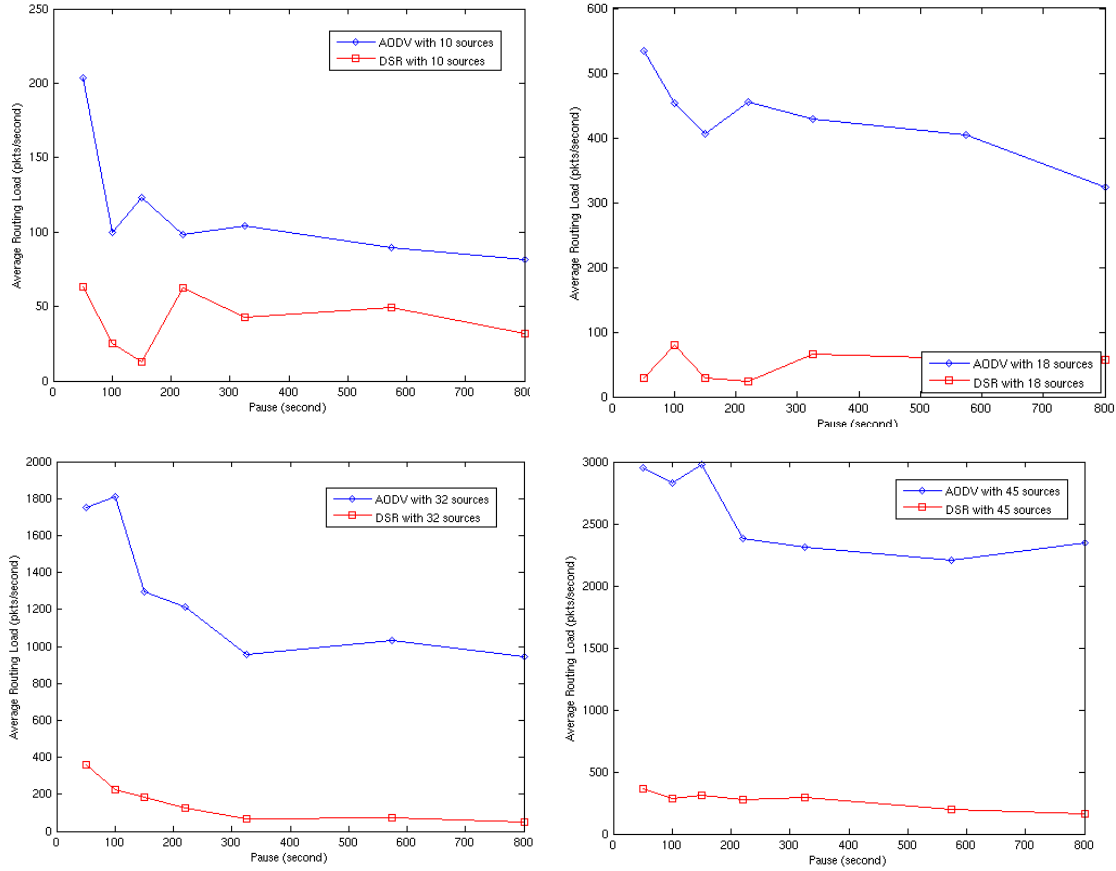
Figure 12 shows about the average delay of AODV and DSR. Since AODV has much more routing packets than DSR, and those routing packets will consume more bandwidth, AODV then will have more delay than DSR.

In this test, under normal load which is 18 sources, AODV cost become higher, this is because routing packets are more with AODV which may cause some latency of sending data packets. When load become heavy which is 32 sources, DSR with small pause time have more delay time since stale routes often be choose, which lost many delivery time. Under heavy load which is 45 sources, AODV become higher again, this is because though many stale routes in DSR were used, AODV also generate many more RREQ pkts, so the

RREQ pkts in AODV is indeed a great bottle neck.

In addition, one interesting observation is that the delays for both protocols increase with 45 sources with very low mobility. This is due to a high level of network congestion and multiple access interferences at certain regions of the ad hoc network. Neither protocol has any mechanism for load balancing, i.e., for choosing routes in such a way that the data traffic can be more evenly distributed in the network. This phenomenon is less visible with higher mobility where traffic automatically gets more evenly distributed due to source movements.

### 5.2.3. Average routing overhead



**Figure 13 Average routing overhead comparison**

Since AODV always has more routing control packets than DSR, the routing overhead of AODV than will always be higher even in stressful environment. Under heavy load, though DSR may incline to choose wrong route, however, under such situation, AODV will also generate far more control packets than DSR.

### 5.3. Simulation conclusion

In particular, DSR uses source routing and route caches and does not depend on any periodic or timer-based activities. DSR exploits caching aggressively and maintains multiple routes per destination. AODV, on the other hand, uses routing tables, one route per destination, and destination sequence numbers, a mechanism to prevent loops and to determine freshness of routes. We used a detailed simulation model to demonstrate the performance characteristics of the two protocols. The general observation from the simulation is that for application oriented metrics such as delay and delivery rate. DSR outperforms AODV in less

stressful situations. AODV, however, outperforms DSR in more stressful situations. DSR, however, consistently generates less routing load than AODV. The poor delay and throughput performances of DSR are mainly attributed to aggressive use of caching, and lack of any mechanism to expire stale routes or to determine the freshness of routes when multiple choices are available. Aggressive caching, however, seems to help DSR at low loads and also keeps its routing load down. We believe that mechanisms to expire routes and/or determine freshness of routes in the route cache will benefit DSR performance significantly.

## 6. Conclusion

The successful test on the comparison of AODV and DSR shows that our performance evaluation mechanism developed by this project is really effective for scalable performance test in NS-2. It also could be easy to use for measure the network routing protocols' performance, meanwhile, since it has the fix model of

analysis the trace file, with some minor modification, it will then be apply to measure other kinds of stuffs with the whole network simulation.

However, since we now only explore some important fields of the trace file, in the future, we still need to provide the measurement with other fields of the trace file and analysis more details on the things what we can get in the trace file.

## 7. References

- [1] The network simulator - ns-2. <http://www.isi.edu/nsnam/ns/>.
- [2] K. Fall and K. Varadhan (Eds.), ns notes and documentation, 1999. <http://wwwmash.cs.berkeley.edu/ns/>.
- [3] NS by Example, <http://nile.wpi.edu/NS/>
- [4] Speeding up NS-2 scheduler. <http://netlab.caltech.edu/~weixl/technical/ns2patch/ns2patch.htm>
- [5] The ns Manual. <http://www.isi.edu/nsnam/ns/doc/index.html>
- [6] S.R. Das, R. Castaneda, J. Yan, and R. Sengupta, "Comparative performance evaluation of routing protocols for mobile, ad hoc networks", In 7th Int. Conf. on Computer Communications and Networks (IC3N), pp: 153–161, October 1998.
- [7] Das, S.R., R. Castaeda and J. Yan, "Simulation-based performance evaluation of routing protocols for mobile ad hoc networks.", Mobile Networks and Applications, 2000, pp: 179-189.
- [8] Introduction to Shell Scripting. <http://www.uwsg.iu.edu/usail/concepts/shell-scripting.html>
- [9] How to write a shell script. [http://vertigo.hsrl.rutgers.edu/ug/shell\\_help.html](http://vertigo.hsrl.rutgers.edu/ug/shell_help.html)
- [10] Steve's Bourne/Bash shell scripting tutorial. <http://steve-parker.org/sh/sh.shtml>
- [11] Shell Programming. <http://www.linuxfocus.org/English/September2001/article216.shtml>
- [12] NS-2 Trace Formats. <http://k-lug.org/~griswold/NS2/ns2-trace-formats.html>
- [13] Introduction to PerlScript. <http://www.cpan.org/authors/id/M/MS/MSERGEANT/PSIntro.html>
- [14] Perl Tutorial. <http://www.comptechdoc.org/independent/web/cgi/perlmanual/>
- [15] Perkins, C.E., E.M. Belding-Royer and S.R. Das, "Ad hoc on-demand distance vector (AODV) routing", IETF Internet Draft. <http://www.ietf.org/internetdrafts/draft-ietfmanet-aodv-11.txt>, 2002.
- [16] Ram Ramanathan and Jason Redi, "A Brief of Overview of Ad Hoc Networks: Challenges and Directions", BBN Technologies.
- [17] Tony Larsson and Nicklas Hedman, "Routing Protocols in Wireless Ad-hoc Networks-A Simulation Study", Stockholm, 1998
- [18] ElizabethMRoyer, Chai-Keong Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks", RFC 2409, IEE Personal Communications , 1999
- [19] H.-W. Cha, J.-S. Park, and H.-J. Kim, "Support of internet connectivity for AODV", IETF Internet Draft, draft-cha-manet-AODV-internet-00.txt, February 2004.
- [20] Broch, J., D.B. Johnson and D.A. Maltz, "The Dynamic Source Routing protocol for mobile Ad Hoc network", Internet draft, Draft-ietf-Manet-dsr-03.Txt, April 2000.
- [21] Johnson, D.B. and D.A. Maltz, "Dynamic source routing in ad hoc wireless networks", Mobile Computing, 1996, pp: 153-181.
- [22] Pekka Savola, "DSR: Dynamic Source Routing", CSC/FUNET, Helsinki University of Technology.
- [23] Garcia-Luna-Aceves, J.J. and M. Spohn, "Source-tree routing in wireless networks", Proceedings of 7th International conference on Network Protocols, 1999.
- [24] Yih-Chun Hu, David B.Johnson, "Caching Strategies in On-Demand Routing Protocols for Wireless Ad Hoc Networks", Carnegie Mellon University, Pittsburgh
- [25] Perkins,C.E., E. M. Royer, S. R. Das and M.K. Marine, "Performance comparison of two on-demand routing protocols for ad hoc networks", IEEE Personal Communications, 2001, pp: 16-28.
- [26] Jiang, H. and J.J. Garcia-Luna-Aceves, "Performance comparison of three routing protocols for ad hoc networks", Proceedings of IEEE ICCCN 2001, pp: 547-554.
- [27] Camp, T., J. Boleng, B. Williams, L. Wilcox and W. Navidi, "Performance Comparison of two location based routing protocols for ad hoc networks", Proceedings of IEEE Infocom 2002, New York.