# Chapter 2

# Wireless Networks - Preliminaries

***This chapter describes an overview and classification of networks used for communication. A comparative simulation study regarding nature of wired and wireless network through commercially available simulators such as: network simulator-2 (NS-2) simulation is discussed.***

---

Though wireless systems have existed since the 1980's it is only in recent times that wireless systems have started to make inroads into all aspects of human life. In the next generation of wireless communication systems, there will be a need for the rapid deployment of independent mobile users. Significant examples include establishing survivable, efficient, dynamic communication for emergency/rescue operations, disaster relief efforts, and military networks. Network scenarios which cannot rely on centralized and organized connectivity can be conceived as applications of **Mobile Ad Hoc Networks**.

A Network is defined as the group of people or systems or organizations who exchange their information for their purpose. Computer networks were started as a necessity for sharing files and printers but after that this has moved from that particular job of file and printer sharing to application sharing and business logic sharing. Tenenbaum [1] defines computer networks as a system for communication between computers. These networks may be fixed (cabled, permanent) or temporary. Wired networks [2] are generally connected with the help of wires and cables. The connection is usually established with the help of physical devices like Switches and Hubs in between to increase the strength of the connection. These networks are usually more efficient, less expensive and much faster than wireless networks [3]. Once the connection is set there is a very little chance of getting disconnected. Wireless communication technology is steadily and rapidly increasing. People wish to use their network terminals (laptops, PDAs, etc.) anywhere and anytime. Wireless connectivity gives users the freedom to move where they desire. There exist numerous different wireless networks varying in the way the nodes interconnect.

Infrastructure-based [4, 5] networks include traditional cellular networks and wireless LANs (with centralized control module). Infrastructures less networks include ad hoc networks and sensor networks. An ad hoc wireless networks or infrastructure less network are defined as the category of wireless networks that utilize multihop radio relaying and are capable of operating without the support of fixed infrastructure. The absence of any central coordinator or base station makes the routing a complex one. In an ad hoc wireless network, the routing and resource management are done in a distributed manner in which all nodes coordinate to enable communication among them [6]. In infrastructure less [6] networks the nodes themselves use each other as routers, so these nodes should be more intelligent than the nodes in centralized networks with APs. There are a lot of situations where infrastructure-less networks are needed: military operations, emergency services, conferencing, game parties, home networking, etc. If the wireless nodes are within the range of each other, the routing is not necessary. **Figure 2.1** depicts representative classification of taxonomy for wireless networks.

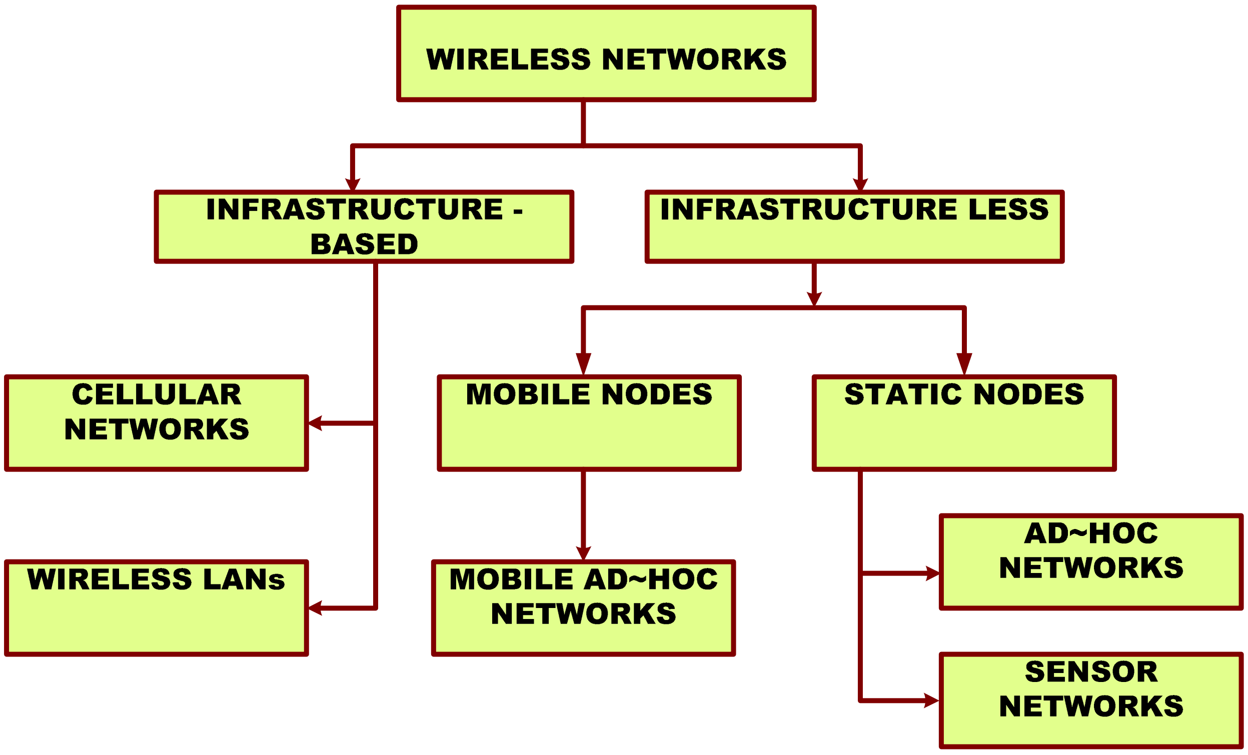


**Figure 2.1:** ***Wireless Networks: Taxonomy.***

## 2.1. Wireless Ad-Hoc Networks - Characteristic & Applications

Characteristics of an infrastructure less wireless network [6] viz.: Wireless ad hoc network (WANET), Vehicular ad hoc networks (VANET) [7], Mobile ad hoc networks (MANET) [8] and Wireless Sensor networks (WSN) [9] are summarized in **Table 2.1.**

| Sr. No. | Characteristics | Purpose/Reason |
|---|---|---|
| 1 | Unreliability of links between nodes | Because of the limited energy supply for the wireless nodes and mobility of the nodes, wireless links between mobile nodes in the ad hoc network are not consistent for communication participants. |
| 2 | Dynamic topology | Due to continuous motion of nodes, the topology of mobile ad hoc network changes constantly, nodes can continuously move |

| | | |
|---|---|---|
| | | into and out of the radio range of other nodes in the ad hoc network, and routing information will be changing all the time because of movement of the nodes. |
| 3 | Shared common broadcast radio channel | Data send by a node is also shared by the malicious node in the channel |
| 4 | In-secured operational environment | Nodes move in and out of hostile and insecure environment. Limited resource availability: Resources like Battery power, bandwidth and computational power are limited in ad-hoc. |

**Table 2.1**: Wireless Ad-Hoc Networks - Characteristic

Generally, ad hoc networks are suitable for all situations in which a temporary communication is desired or in case the building of network infrastructure not possible. There are many applications to ad hoc networks, such as home networks, group discussions, vehicle communications, Military or police exercises, Disaster relief operations, Mine site operations, Urgent Business meetings, Robot data acquisition.

- **Emergency services:** Search and rescue operations; Disaster recovery; Replacement of a fixed infrastructure in case of environmental disasters**;** Policing and fire fighting ; Networks of visitors at airports.
- **Home and enterprise networking:** Home/Office Wireless Networking (WLAN); Personal Area Network (PAN); Conferences, meeting rooms; Networks at construction sites.
- **Educational applications:** Setup virtual classrooms or conference rooms. Setup ad hoc communication during conferences, meetings or lectures; Universities and campus settings.
- **Entertainment:** Multi-user games; Robotic pets and Theme parks.

## 2.2 Network Simulators & Simulations – State of Art

In wireless networks the nodes communication between each other is unstable due to mobility of the wireless nodes and the quality of the wireless links is fluctuating heavily. Wireless nodes in these types of network are generally portable, so it is not feasible to implement algorithms with a large processing power or memory footprint. Therefore, it is inevitable to thoroughly test protocols to be able to assess their performance in the anticipated application scenario.

Area of research in wireless networking depends on the capabilities of simulation tools [10, 11] and, more specifically, on the scalability of wireless network simulators [12]. Analytically quantifying the performance and complex behavior of even simple protocols in the aggregate is often imprecise. Furthermore, performing actual experiments is onerous: acquiring hundreds of devices, managing their software and configuration, controlling a distributed experiment and aggregating the data, possibly moving the devices around, finding the physical space for such an experiment, isolating it from interference are but some of the difficulties that make empirical endeavors daunting.

Prevailing trends are...

- **TESTBED IMPLEMENTATION**: Developed concepts are implemented and deployed on actual hardware. While test beds might yield the most accurate results, there are several drawbacks, such as: the need to obtain hardware and the severely limited monitoring and debugging possibilities, as well as, high effort needed to create an artificial environment resembling the real application scenario. Hence, testbed implementations will generally be an option only for smaller numbers of nodes and during the later stages of the implementation phase.
- **SIMULATION**: The physical world is modeled in wireless network simulation software. This avoids the issues with testbed implementations explained above. On the other hand, models can capture reality only to a limited extent, which implies that simulation results will generally not be as accurate as real implementations.

To evaluate the behavior and performance of wireless networks, simulations are a good compromise between cost and complexity, and accuracy of the results. Since there are many simulators for wireless networks available but, it's difficult to decide which simulator use for which application [13, 14]. Section describes the features of the simulators, their strengths and weaknesses. The effort and procedure necessary for installation, familiarization, and implementation (needed lines of code and lines for configuration) and visualization are also discussed.

### 2.2.1 OPNET

OPNET Modeler [15] is a software product for the simulation and modeling of communication protocols and Internet technologies. However, OPNET's considerable amounts of source code and supporting application programming interfaces (API) can be quite overwhelming, even for experienced developers. This paper attempts to demystify the process of modeling in OPNET and enumerate the key steps for creating new simulation models using the OPNET Modeler software package.

OPNET [16] uses a relatively simple GUI to interface its rather very complex kernal engine. It provides a comprehensive framework for modeling wired as well as wireless network scenarios. Simulation models are organized in a hierarchy consisting of three main levels: the simulation network, node models and process models. The top level refers to the simulation scenario or simulation network. It defines the network layout, the nodes and the configuration of attributes of the nodes comprising the scenario. The node models are at the second level in the hierarchy and consist of an organized set of modules describing the various functions of the node. The modules in the nodes are implemented using process models, the lowest level in the hierarchy. Process models consist of finite state machines, definitions of model functions, and a process interface that defines the parameters for interfacing with other process models and configuring attributes. Finite state machine models are implemented using Proto C, which is a discrete event library based on C functions. The hierarchal structure of the models, coupled with support for C language programming, allows for easy development of communication or networking models.

[17] Explains model architecture of OPNET simulator in general and MANET models. It also addresses how to configure AODV parameters in OPNET and how to obtain the results of AODV using this simulator. [18] Has use the OPNET simulation tool for modeling and analysis of packet data networks is described. In this simulation of two types of high-performance networks: Fiber Distributed Data Interface and Asynchronous Transfer Mode. [19] Has implemented numerous simulations of MANET using OPNET as the primary tool with the goal of achieving real-time simulations while maintaining or increasing the fidelity. One of the primary techniques for achieving real-time results is to run the simulations concurrently in a multithreaded system, which is supported by OPNET's parallel kernel.

### 2.2.2 GLOMOSIM

GloMoSim is a scalable simulation *library* designed at UCLA Computing Laboratory to support studies of *large-scale network* models, up to millions of nodes, using *parallel* and/or distributed execution on a diverse set of parallel computers (with both distributed and shared memory). It has been designed for wireless and wired networks systems, but it currently supports only protocols for purely *wireless* networks. GloMoSim is a library for the C-based parallel discrete-event simulation language *PARSEC* [20]. One of the important distinguishing features of PARSEC is its ability to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. PARSEC is designed to cleanly separate the description of a simulation model from the underlying simulation protocol, sequential or parallel, used to execute it.

GloMoSim implements a technique called “node aggregation,” wherein the states of multiple simulation nodes are multiplexed within a single Parsec entity. While this effectively reduces memory consumption, it incurs a performance overhead and also increases code complexity. Moreover, the aggregation of state also renders the look ahead techniques are impractical, as has been noted by the authors. GloMoSim has been shown to scale to 10,000 nodes on large, multi-processor machines.

### 2.2.3 OMNET++

OMNeT++ [21] is an object-oriented modular discrete event network simulation framework. It has a generic architecture, so it can be (and has been) used in various problem domains: modeling of wired and wireless communication networks ,protocol modeling ,modeling of queuing networks ,modeling of multiprocessors and other distributed hardware systems ,validating of hardware architectures ,evaluating performance aspects of complex software systems .

OMNeT++ itself is not a simulator of anything concrete, but rather provides infrastructure and tools for *writing* simulations. One of the fundamental ingredients of this infrastructure is component architecture for simulation models. Models are assembled from reusable components termed *modules*.

Modules can be connected with each other via gates (other systems would call them ports), and combined to form compound modules. The depth of module nesting is not limited. Modules communicate through message passing, where messages may carry arbitrary data structures. Modules can pass messages along predefined paths via gates and connections, or directly to their destination; the latter is useful for wireless simulations, for example. Modules may have parameters that can be used to customize module behavior and/or to parameterize the model's topology. Modules at the lowest level of the module hierarchy are called simple modules, and they encapsulate model behavior. Simple modules are programmed in C++, and make use of the simulation library. In general, it can be used for the modeling and simulation of any system where the discrete events approach is suitable, and which can be conveniently mapped into entities communicating by exchanging messages.

[22] Has described the adaptation of diverse implementations of ad hoc routing protocols so they can be incorporated in OMNeT++ INET library. The adaptation has been focused on the protocols AODV, DSR and DYMO, which are among the most employed routing schemes in studies and prototypes concerning MANETs [23]. The developed modules have permitted to integrate in powerful and popular simulation tool existing and validated software that is presently performing in actual implementations of ad hoc networks. Therefore, the simulations with OMNeT++ could be compared even with the results obtained with real MANET test beds.

### 2.2.4 QUALNET

The QualNet Developer IDE is a GUI based program for developing network scenarios that comes with QualNet 5.0 [24]. It can be used to visually design network scenarios and then run simulations of these networks. Although networks can be designed and simulated in a command-line fashion as well, on the Developer IDE package.

QualNet is a comprehensive suite of tools for modeling large wired and wireless networks. It uses simulation and emulation to predict the behavior and performance of networks to improve their design, operation and management. QualNet enables users to: Design new protocol models, Optimize new and existing models, Design large wired and wireless networks using pre-configured or user-designed models, Analyze the performance of networks and perform what-if analysis to optimize them [24].

The kernel of QualNet is a, SNT-proprietary, parallel discrete-event scheduler. It provides the scalability and portability to run hundreds and thousands of nodes with high-fidelity models on a variety of platforms, from laptops and desktops to high performance computing systems. Users do not directly interact with the kernel, but use the QualNet API to develop their protocol models.

QualNet includes support for a number of model libraries that enable you to design networks using SNT-developed protocol models. Purchase of QualNet includes the Developer Model Library; additional libraries for modeling WiFi networks, mobile ad-hoc networks (MANET), military radios, WiMAX and cellular models are also available.

An alternative approach to network testing and evaluation based on the real time emulation capability of the QualNet simulator is discussed in [25]. It provides an overview of the approach and provides illustrations to outline how a testbed based on real-time emulation can be set up. The effectiveness of such a testbed is highlighted by a use case, where performance evaluation results are presented for a wireless ad hoc network which is to be deployed over a range of scenarios. [26], proposes a receiver-centric approach for location tracking and MAC protocol. In order to track the location of its neighbor, each node n periodically *collects* its neighborhood information and forms an Angle- Signal Table (AST). Based on AST, a node n knows the direction of node m and controls the *medium access* during transmission-reception. The performance evaluation on QualNet network simulator indicates that protocol is highly efficient with increasing number of communications and with increase in data rate [24].

### 2.2.5 MATLAB

MATLAB/SIMULINK is one of the most successful software packages currently available, and is particularly suited for work in control [28]. It is a powerful, comprehensive and user-friendly software package for simulation studies. Our objective here is to help the reader gain a basic understanding of this software package by showing how to set up and solve a simulation problem. Interested readers are encouraged to further explore this very complete and versatile mathematical computation package. **SIMULINK** is a software package for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multirate, i.e., have different parts that are sampled or updated at different rates. SIMULINK toolboxes are built using MATLAB. They are specialized collection of m-files for working on particular class of problem. We can inspect m-files, add to them, or use them for templates when creating our own functions. Every toolbox is available on any computer platform that runs MATLAB. Every toolbox builds on the robust numeric, rock-solid accuracy.

#### 2.2.5.1 TRUETIME Toolbox[1]

TRUETIME [29] currently supports Matlab 7.x (R14 and later) with Simulink 6.x and Matlab 6.5.1 (R13.1) with Simulink 5.1. Support for Matlab 6.1 (R12.1) with Simulink 4.1 was dropped in TRUETIME 1.3. A C++ compiler is required to run TRUETIME in the C++ version. For the Matlab version, pre-compiled files are provided in the archive that is downloaded from the TRUETIME web site. The following compilers are currently supported (it may, of course, also work using other compilers):

- Visual Studio C++ 7.0 (for all supported Matlab versions) for Windows
- gcc, g++ - GNU project C and C++ Compiler for LINUX and UNIX

Before starting MATLAB, environment variable TTKERNEL has to be set to point to the directory with the TRUETIME kernel files, $DIR/kernel. Procedure steps for setting the environment variable are…..

1. For Unix/Linux: export TTKERNEL=$DIR/kernel

   OR

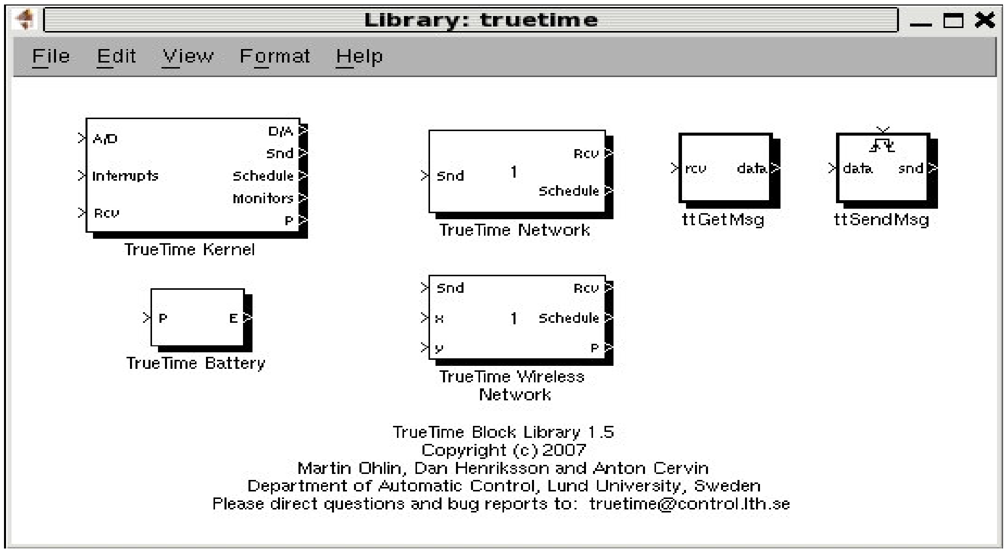
1. Windows: use Control Panel / System / Advanced / Environment Variables
2. Then add the following lines to your Matlab startup script. This will set up all necessary paths to the TRUETIME kernel files.

```
addpath ([getenv ('TTKERNEL')])
init_truetime
```

3. Start MATLAB and issue the command >> truetime.

TRUETIME block library, shown in **Figure 2.2.**

[1] Presented a Paper **"Simulation of Reactive Protocol Using Matlab/Truetime"** at a **State Level Paper Contest** called "Wireless Technologies in Automation and Communication **: WTAC-2010**" held at Institution of Engineering(India),Vasvik Bhavan,Vadodara on 10th January, 2010 organized by IETE Vadodara.



TrueTime Block Library 1.5
Copyright (c) 2007
Martin Ohlin, Dan Henriksson and Anton Cervin
Department of Automatic Control, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se

**Figure 2.2: TRUETIME library block**

TRUETIME archive contains pre-compiled files, no compilation is required to run TRUETIME with the M-file API.

TRUETIME supports simulations written in C++ code, but needs to be compiled, which requires configuration of C++ compiler in MATLAB by issuing the command >> mex –setup

TRUETIME blocks are connected with ordinary Simulink blocks to form a real time control system. Before a simulation can be run, however, it is necessary to initialize kernel blocks and network blocks, and to create tasks, interrupt handlers, timers, events, monitors, etc for the simulation. [30] Gives a brief introduction to the TrueTime simulator and then gives several examples on how TrueTime can be used to simulate networked control systems. Among the examples are time-triggered and event-based networked control and AODV routing in wireless ad-hoc networks.

### 2.2.6 NETWORK SIMULATOR- NS2

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkeley written in C++ and OTcl. NS is basically useful for simulating wired and wireless networks. It implements 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 more. It also implements multicasting and some of the MAC layer protocols for LAN simulations.

Ns-2 [31] is a packet-level simulator which is essentially a centralized discrete event scheduler to schedule the events such as packet and timer expiration. The centralized event scheduler cannot accurately emulate "events occurred at the same time" instead; it can only handle events occurred one by one in time. However, this is not a serious problem in most network simulations, because the events here are often transitory. Besides, ns-2 implements a variety of network components and protocols. Notably, the wireless extension, derived from CMU Monarch Project [32], has 2 assumptions simplifying the physical world:

(1) Nodes do not move significantly over the length of time they transmit or receive a packet. This assumption holds only for mobile nodes of high-rate and low-speed. Consider a node with the sending rate of 10Kbps and moving speed of 10m/s, during its receiving a packet of 1500B, the node moves 12m. Thus, the surrounding can change significantly and cause reception failure.

(2) Node velocity is insignificant compared to the speed of light. In particular, none of the provided propagation models include Doppler effects, although they could.

The NS project is now a part of the VINT project that develops tools for simulation results display, analysis and converters that convert network topologies generated by well-known generators to NS formats.

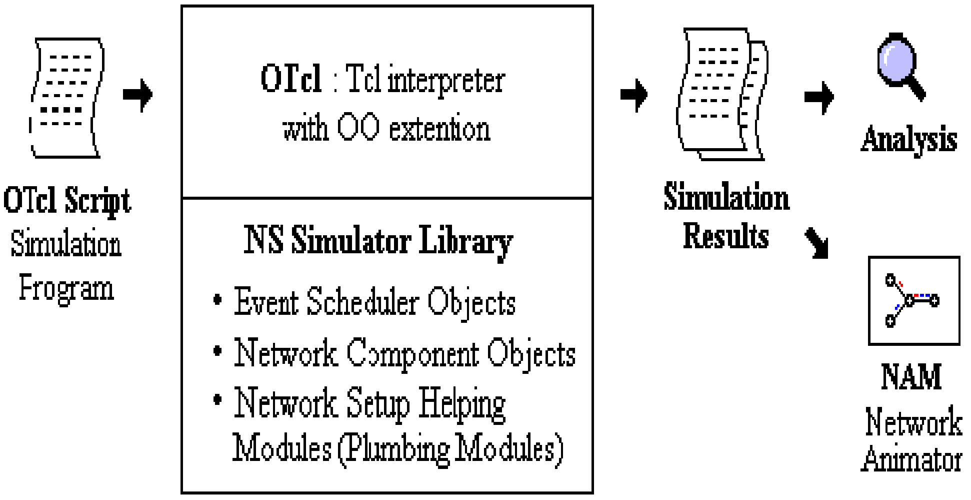


**Figure 2.3: Simplified User's View of NS**

As shown in **Figure 2.3**, in a simplified user's view, NS is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object). To setup and run a simulation network, a user should write an OTcl script that 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. In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with packet pointed by the event. Network components communicate with one another passing packet. All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet.

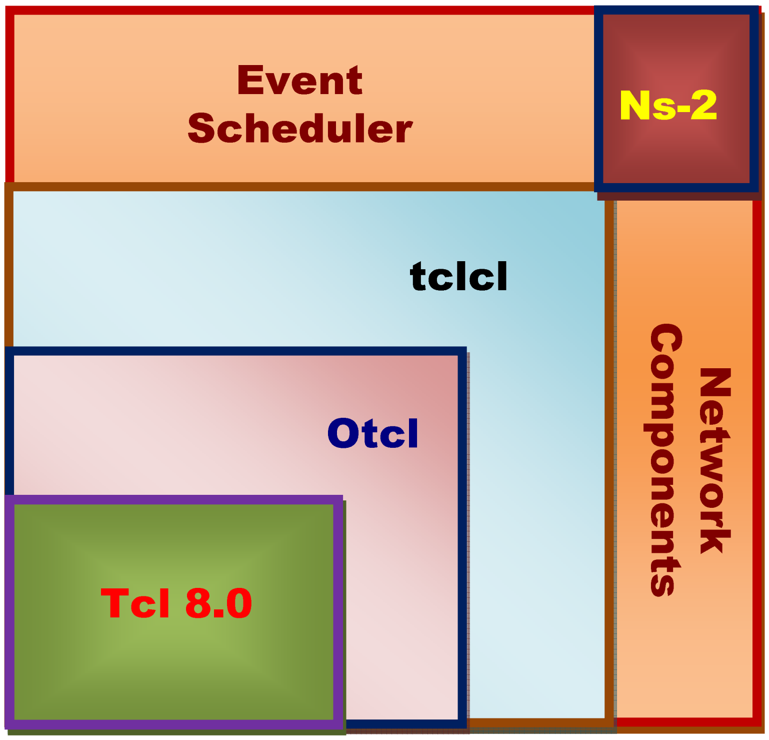


**Figure 2.4:** Architectural View of NS

**Figure 2.4** shows the general architecture of NS. In this figure a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is an OO extended Tcl interpreter with network simulator libraries.

When a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, if specified to do so in the input Tcl (or more specifically, OTcl) script. The data can be used for simulation analysis as an input to a graphical simulation display tool called Network Animator (NAM) that is developed as a part of VINT project. NAM has a nice graphical user interface has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis. The behavior and performance evaluation of routing protocols for wireless networks, simulations are a good compromise between cost and complexity, on the one hand, and accuracy of the results, on the other hand. Since there are many simulators for wireless networks, it is often difficult to decide which simulator to choose.

[33] Present a case study in which four popular wireless network simulators were used to evaluate a well-known topology control protocol (SPAN). They describe outstanding and desirable but missing features of the simulators, outlining their strengths and weaknesses. Also, the comparison of the amount of effort needed for installation, familiarization, and implementation (needed lines of code and lines for configuration) and visualization. [34] The characteristics of simulation environments for telecommunications networks, and, more specifically, for mobile ad hoc networks are discussed. Different available network simulators are discussed and compared according to a number of criteria ranging from the reliability of the provided simulation models, to the degree of usability of software and graphical interfaces. In this the simulation environment presented which is the most appropriate to run experiments to test and validate the design of novel adaptive routing algorithms developed in the framework of BISON. The document concludes that the (commercial) QualNet simulator [35] appears to be the most satisfactory product currently available. In [36] different network modeling and simulation (MS) tools are presented. These network MS tools can be used in education and research as well as practical purposes. They vary with their characteristics. This paper reviews some of the most important network MS tools developed recently. This paper also shows a classification of the tools used in communications networks. [37] Compares protocols and models implemented in SWANS to the corresponding implementations in ns-2. Using identical input parameters, they show where results are comparable and analyze reasons for differences. By showing that results achieved with JiST/SWANS are equivalent to those of ns-2, it supports the usage of JiST/SWANS. As ns-2 performance problems when simulating hundreds or thousands of nodes and the complex mixture of Tcl and C/C++ code in ns-2, JiST/SWANS could be an interesting alternative. [38] Introduce simulation execution framework for a further simplification when using JiST/SWANS, which allows for generation, execution and evaluation of complex simulation studies.

Because of their high cost and their lack of flexibility of networks, experimentation is mostly achievable through simulation. Numerous tools exist for MANETs simulation, including ns-2 and GloMoSim which are the two most popular ones. [39] Provides a State of the Art of MANETs simulators and associated simulation techniques. First it gives an overview of the domain. Then it provides a map of the main characteristics that MANETs simulation tools should feature and the current support of these. Finally, a description for each simulator is provided, including an explanation of what make them appealing solutions.

## 2.3 Network Simulation using NS-2

[40] Addresses simulations of Mobile Ad-Hoc Network (MANET) with the simulation tool Network Simulator NS-2. Since the simulation software does not include advanced protocols and sophisticated mobile network nodes new implementations have been performed. Firstly, a new mobile gateway has been introduced and secondly a standardized routing protocol has been enhanced to manage interaction between standard mobile nodes and the newly implemented gateway. Test simulations show that all modifications work properly and results are reasonable. A simulation model with MAC and physical layer models is used to study interlayer interactions and their performance implications. A variety of workload and scenarios, as characterized by mobility, load and size of the ad hoc network is discussed in [41]. The performance differentials are analyzed using varying network load, mobility, and network size. Simulations carried out based on the Rice Monarch Project that has made substantial extensions to the ns -2 network simulator to run ad hoc simulations.

### 2.3.1 Steps to use NS-2 and creating trace file

To use the Network Simulator NS-2 for the simulation of wired and wireless network first the tcl file is required which describes the topology of the network and communication between the nodes present in the network [28].

#### 2.3.1.1 Creation of tcl file

Following are the steps 1 to 12 for generating the trace .tr file and .nam file for checking the wireless network performance with different parameters.

- Create an instance of the simulator:
- Setup trace support by opening
- Next create a topology object
- Create the object God, "God (General Operations Director)
- Configuring the nodes
- Create nodes and the random-motion for nodes is disabled
- Give nodes positions to start with
- Setup node movement as the following example

- Setup traffic flow between the two nodes as follows: TCP connections between nodes
- Define stop time when the simulation ends and tell nodes to reset which actually resets their internal network components.
- Finally the generated file for any particular protocol is run and the output files are generated namely out.tr and out.nam and out.tcl.

### 2.3.1.2 Network Animator (NAM) Viewer

After finishing of writing tcl file for the network we can visualize the packet communication between the nodes participating in the network when the tcl file is executed. It also generate the trace file which contains the all the data related to packet communication between the nodes.

### 2.3.1.3 Trace file format

In “*.tr “file each line consists of following information as shown in **Figure 2.5**

| Event | Time | From node | To node | Packet type | Packet Size | Flags | Fid | Src addr | Dest addr | Seq number | Pkt id |
|---|---|---|---|---|---|---|---|---|---|---|---|

| | |
|---|---|
| r : receive (at to_node) | d: drop (at queue) |
| + : enqueue (at queue) | Src_addr: node.port( 3.0) |
| - : dequeue (at queue) | Dest_addr: node.port(0.0) |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| r | 1.3556 | 3 | 2 | ack | 40 | ------- | 1 | 3.0 | 0.0 | 15 | 201 |
| + | 1.3566 | 2 | 0 | ack | 40 | ------- | 1 | 3.0 | 0.0 | 15 | 201 |
| - | 1.3566 | 2 | 0 | ack | 40 | ------- | 1 | 3.0 | 0.0 | 15 | 201 |
| r | 1.35576 | 0 | 2 | tcp | 1000 | ------- | 1 | 0.0 | 3.0 | 29 | 199 |
| + | 1.35576 | 2 | 3 | tcp | 1000 | ------- | 1 | 0.0 | 3.0 | 29 | 199 |
| d | 1.35576 | 2 | 3 | tcp | 1000 | ------- | 1 | 0.0 | 3.0 | 29 | 199 |
| + | 1.356 | 1 | 2 | cbr | 1000 | ------- | 2 | 1.0 | 31 | 157 | 207 |
| - | 1.356 | 1 | 2 | cbr | 1000 | ------- | 2 | 1.0 | 31 | 157 | 207 |

**Figure 2.5: TRACE files Format**

Each line consists of:

- Event Descriptor (+, -, d, r)
- Simulation time (in seconds) of that event
- *From* Node & *To* Node, which identify the link on which the event occurred
- Packet type
- Packet size
- Flags (appeared as "------" since no flag is set). Currently, NS implements only the Explicit Congestion Notification (ECN) bit, and the remaining bits are not used.

- Flow id (fid)
- Source and destination address in forms of "node.port".
- The network layer protocol's packet sequence number. *What about UDP?*
- The last field shows the unique id of the packet.

Each trace line starts with an event (+, -, d, r) descriptor followed by the simulation time (in seconds) of that event, and from and to node, which identify the link on which the event occurred. The next information in the line before flags (appeared as "------" since no flag is set) is packet type and size (in Bytes). Currently, NS implements only the Explicit Congestion Notification (ECN) bit, and the remaining bits are not used. The next field is flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script. Even though fid field may not use in a simulation, users can use this field for analysis purposes. The fid field is also used when specifying stream color for the NAM display. The next two fields are source and destination address in forms of "node**.**port". The next field shows the network layer protocol's packet sequence number. Note that even though UDP implementations do not use sequence number, NS keeps track of UDP packet sequence number for analysis purposes. The last field shows the unique id of the packet.

### 2.3.1.4 Extraction of necessary data from Trace file

We need to extract the necessary data out of this file. GREP is a **UNIX** command to filter a file. The name comes from "search **G**lobally for Lines Matching the **R**egular **E**xpression and **P**rint them". It takes a regular expression on the command line, reads the standard input or list of files, and outputs the lines containing matches for the regular expression.

**Example:** In the trace file, if you are interested only in data concerning tcp packets that went from node 0 to node 2 than,

**grep** "0 2 tcp" tr1.tr > tr2.tr

In the trace file, if you are interested only in the received packets

**grep** "^r" tr1.tr > tr2.tr

Another option is to use AWK file. **AWK** is a general purpose *computer language.* **AWK** is designed for processing text-based data, either in files or data streams. The name **AWK** is derived from the surnames of its authors — Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan.

To run AWK file following commands are needed:

```
awk –f file1.awk file2.txt
awk –f file1.awk file2.txt > out.txt
```

Where,
file1.awk : is a command file
file2.txt : is a primary input file
out.txt : is an output file

A typical AWK program consists of a series of lines; each of them is on the form

**/pattern/ {action}**

Pattern is a regular expression Action is a command.

● Most implementations of AWK use extended regular expressions by default.

● AWK looks through the input file; when it finds a line that matches pattern, it executes the command(s) specified in action.

- *awk* or *perl* scripting language is used to interpret the trace file, it extract data from trace file trace.tr.
- Write the awk script (that is similar to C language) and save it in *****.awk:
- Following is the sample of awk file written for calculating packet delivery ratio for the trace file for wireless network example.

**# awk file to find packet delivery ratio.**

```
# ==============================pdr.awk =======================

    BEGIN {highest_packet_id = 0 ;}
    {
     action = $1;
     time = $2;
     node =$3;
     tracename = $4;
     type = $7;
     packet_id =$6;
     if (packet_id > highest_packet_id)
       highest_packet_id = packet_id;
     if (action == "s"){
     if (type == "cbr" && tracename == "AGT"){
        send_time[packet_id] = time;}
    #else {send_time[packet_id] = 0;}
    }
    else if (action == "r"){
        if (type == "cbr" && tracename =="AGT"){
           rcv_time[packet_id] = time;
        }
    #else{rcv_time[packet_id] = 0;}
     }
    }
     END {packet_no = 0; total_delay = 0;
           for (packet_id = 0; packet_id <=highest_packet_id; packet_id++){
                if ((send_time[packet_id]!=0) && (rcv_time[packet_id]!=0)){
                    start = send_time[packet_id];
                    end = rcv_time[packet_id];
                   packet_duration = end-start;}
    else
           packet_duration = -1;
    if (packet_duration > 0)
    {packet_no++;
      total_delay = total_delay + packet_duration; }
    }
       printf("%d %f %f\n", packet_no, total_delay, total_delay/packet_no);
    }
```

## 2.3.2 Simulation of wired Networks

To study the behavior of Wired Network we have created the network (**Figure 2.6(a)**) with six wireless nodes behaves like mobile nodes in the network area. **Figure 2.6(b)** shows the situation of the network where the queue gets overflow and the packets drops.

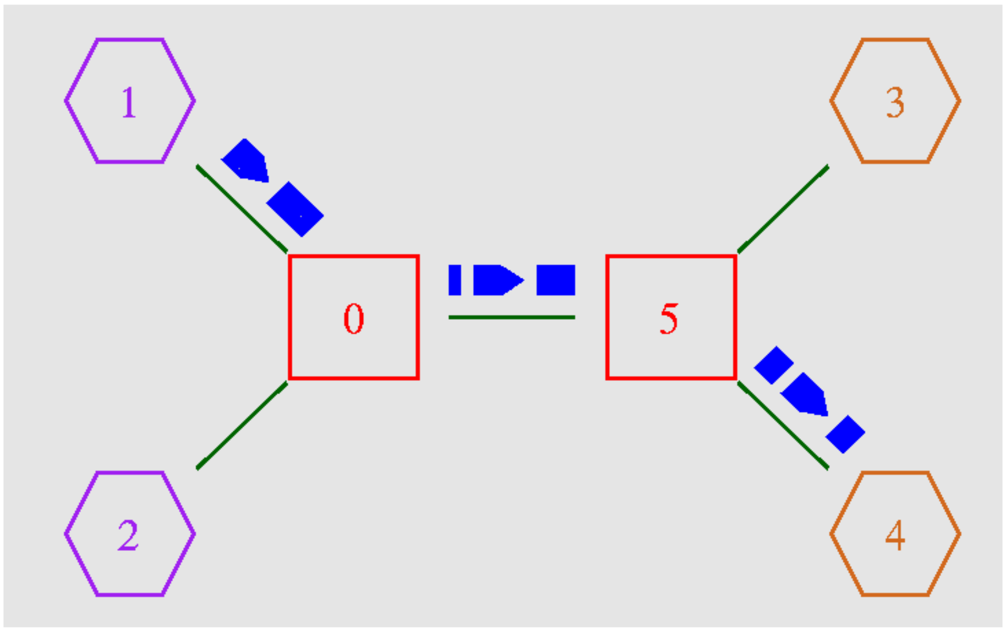


**Figure 2.6(a): Topology of the network in NAM**

```
# mynode5.tcl ---prepared by Dharmistha on September 15, 2009
#create a simulator object
set ns [new Simulator]
#open a file for writing that is going to be used for the nam trace data
#this line opens the file 'out.nam' for writing and gives it the file handle 'nf'.
set f [open out.tr w]
$ns trace-all $f
set nf [open out.nam w]
$ns namtrace-all $nf
#define the two nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
#this lines will define the color of the nodes.
$n0 color red
$n1 color purple
$n2 color purple
$n3 color chocolate
$n4 color chocolate
$n5 color red
#this lines will define the shape of the nodes.
$n0 shape box
$n1 shape hexagon
$n2 shape hexagon
$n3 shape hexagon
$n4 shape hexagon
$n5 shape box
#creation of the duplex link between the two nodes
#with the queue type droptail and the link with the bandwidth of 1mb and delay of 10ms
#line tells the simulator object to connect the nodes n0 and n1
```

```
# With a duplex link with the bandwidth 1Megabit, a
#delay of 10ms and a DropTail queue.
$ns duplex-link $n1 $n0 1Mb 10ms DropTail
$ns duplex-link $n2 $n0 1Mb 10ms DropTail
$ns duplex-link $n0 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n3 1Mb 10ms DropTail
$ns duplex-link $n5 $n4 1Mb 10ms DropTail
#this line will tell the direction of the duplex link between the nodes no and n1.
$ns duplex-link-op $n1 $n0 orient right-down
$ns duplex-link-op $n2 $n0 orient right-up
$ns duplex-link-op $n0 $n5 orient right
$ns duplex-link-op $n5 $n3 orient right-up
$ns duplex-link-op $n5 $n4 orient right-down
#this line will define the color of the duplex link
$ns duplex-link-op $n1 $n0 color "darkgreen"
$ns duplex-link-op $n2 $n0 color "darkgreen"
$ns duplex-link-op $n0 $n5 color "darkgreen"
$ns duplex-link-op $n5 $n3 color "darkgreen"
$ns duplex-link-op $n5 $n4 color "darkgreen"
###These lines create a UDP agent and attach it to the node n1, then attach a CBR traffic to the
##UDP agent.
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
###These lines create a UDP agent and attach it to the node n1, then attach a CBR traffic to the
##UDP agent.
#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$ns attach-agent $n2 $udp1
#this lines will define the class for the different agents
#second line will define the different colors to the different classes.
$udp0 set class_ 1
$ns color 1 Blue
#this lines will define the class for the different agents
```

```
#second line will define the different colors to the different classes.
$udp1 set class_ 2
$ns color 2 orange
## Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
## Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
#create a Null agent which acts as traffic sink and attach it to node n3.
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
#create a Null agent which acts as traffic sink and attach it to node n4.
set null1 [new Agent/Null]
$ns attach-agent $n4 $null1
#two agents have to be connected with each other.
$ns connect $udp0 $null1
$ns connect $udp1 $null0
#this line tells to the CBR agent when to send data and when to stop sending.
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 1.0 "$cbr1 start"
$ns at 4.5 "$cbr1 stop"
#adds a 'finish' procedure that closes the trace file and starts nam.
proc finish {} {
global ns nf
#$ns flush-trace
close $nf
exec nam out.nam &
exit 0
}
#line tells the simulator object to execute the 'finish' procedure after 5.0 seconds of simulation
time.
$ns at 5.0 "finish"
#finally starts the simulation
$ns run
```

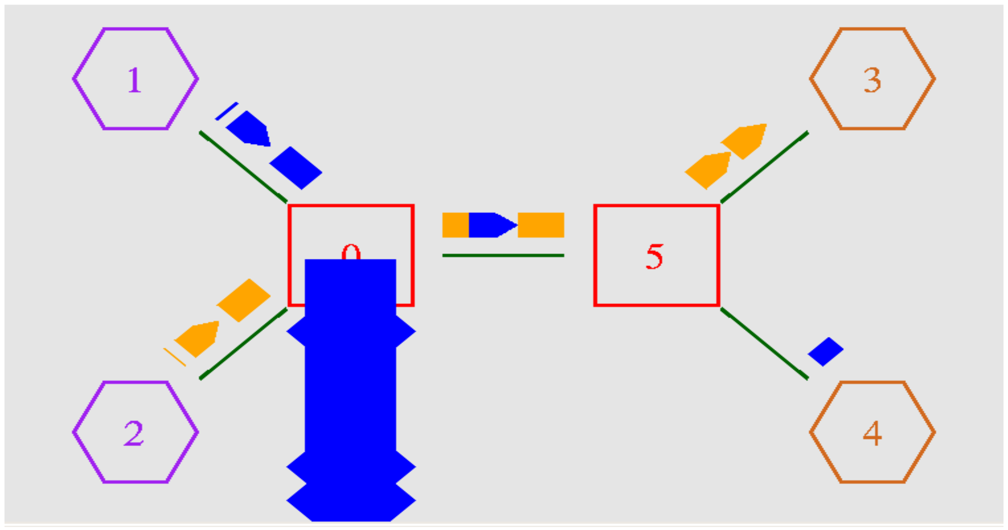


**Figure: 2.6(b): Packet drops in queue of network (in NAM)**

## 2.3.3 Simulation of wireless Network using NS-2

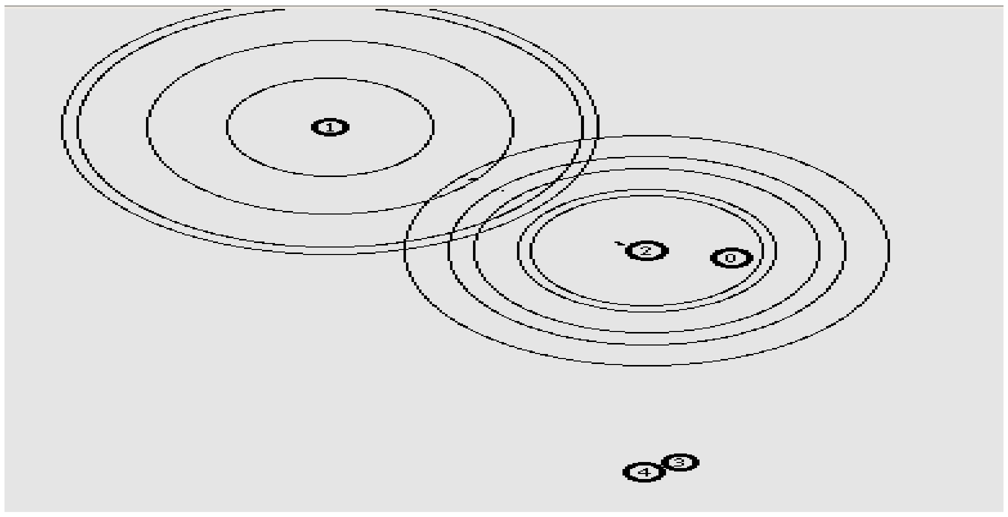


**Figure: 2.7: 5 wireless node topology in NS2**

**Figure 2.7** shows the topology of the wireless network which consists of 5 wireless nodes communication with each other. The circles around nodes show the coverage of transmission of the nodes. Following is the code written in NS2 for the above topology.

```
#   wirelesstest2.tcl --prepared by Dharmistha on September 20, 2009
#
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/802_11                 ;# 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(nn)           5                          ;# number of mobile nodes
set val(rp)           AODV                       ;# routing protocol
#set val(rp)          DSR                        ;# routing protocol
set val(x)                       670
set val(y)                       670

# Initialize Global Variables
set ns_           [new Simulator]
set tracefd    [open wirelesstest2.tr w]
$ns_ trace-all $tracefd

set namtrace [open wirelesstest2.nam w]
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo       [new Topography]

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

# Create God
create-god $val(nn)

# New API to config node:
# 1. Create channel (or multiple-channels);
# 2. Specify channel in node-config (instead of channel Type);
# 3. Create nodes for simulations.

# Create channel #1 and #2
set chan_1_ [new $val(chan)]
set chan_2_ [new $val(chan)]
set chan_3_ [new $val(chan)]
set chan_4_ [new $val(chan)]
set chan_5_ [new $val(chan)]
```

```
# Create node(0) "attached" to channel #1

# configure node, please note the change below.
$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 ON \
                 -macTrace ON \
                 -movementTrace OFF \
                 -channel $chan_1_

set node_(0) [$ns_ node]
set node_(1) [$ns_ node]
set node_(2) [$ns_ node]
set node_(3) [$ns_ node]
set node_(4) [$ns_ node]

$node_(0) random-motion 0
$node_(1) random-motion 0
$node_(2) random-motion 0
$node_(3) random-motion 0
$node_(4) random-motion 0

for {set i 0} {$i < $val(nn)} {incr i} {
        $ns_ initial_node_pos $node_($i) 20
}

#
# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes
#

#$node_(0) set X_ 5.0
#$node_(0) set Y_ 2.0
#$node_(0) set Z_ 0.0

#$node_(1) set X_ 8.0
#$node_(1) set Y_ 5.0
#$node_(1) set Z_ 0.0

$node_(0) set X_ 250.159448320886
$node_(0) set Y_ 320.107989080168
$node_(0) set Z_ 0.000000000000
```

```
$node_(1) set X_ 512.514473960930
$node_(1) set Y_ 475.755796386780
$node_(1) set Z_ 0.000000000000

$node_(2) set X_ 501.042844720300
$node_(2) set Y_ 569.731243003695
$node_(2) set Z_ 0.000000000000

$node_(3) set X_ 332.327451652222
$node_(3) set Y_ 81.202292764558
$node_(3) set Z_ 0.000000000000

$node_(4) set X_ 219.839231807235
$node_(4) set Y_ 174.841925336524
$node_(4) set Z_ 0.000000000000

#Turning off all traffic node first
#$ns_ at 0 "$node_(0) shutdown"
#$ns_ at 0 "$node_(1) shutdown"
#$ns_ at 0 "$node_(2) shutdown"
#$ns_ at 0 "$node_(3) shutdown"
#$ns_ at 0 "$node_(4) shutdown"

#
# Now produce some simple node movements
# Node_(1) then starts to move away from node_(0)
#$ns_ at 20.0 "$node_(1) setdest 490.0 480.0 30.0"
$ns_ at 10.0 "$node_(0) setdest 412.8 392.7 11.5"
$ns_ at 20.0 "$node_(1) setdest 171.6 556.4 15.0"
$ns_ at 30.0 "$node_(2) setdest 274.341843721105 299.239024275597 6.731226074495"
$ns_ at 40.0 "$node_(3) setdest 662.745412787246 470.917833530129 3.340205607452"
$ns_ at 50.0 "$node_(4) setdest 495.488755763193 75.259676974606 12.272713794783"

# Setup traffic flow between nodes
# 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
$ns_ at 40.0 "$ftp start"

#
# 0 connecting to 1 at time 0
#
#$ns_ at 0 "$node_(0) startup"
```

```
#$ns_ at 0 "$node_(1) startup"
#set udp_(0) [new Agent/UDP]
#$ns_ attach-agent $node_(0) $udp_(0)
#set null_(0) [new Agent/Null]
#$ns_ attach-agent $node_(1) $null_(0)
#set cbr_(0) [new Application/Traffic/CBR]
#$cbr_(0) set packetSize_ 512
#$cbr_(0) set rate_ 81920
#$cbr_(0) set random_ 1
#$cbr_(0) set maxpkts_ 10000
#$cbr_(0) attach-agent $udp_(0)
#$ns_ connect $udp_(0) $null_(0)
#$ns_ at 90 "$cbr_(0) start"
#$ns_ at 90 "$cbr_(0) stop"
#$ns_ at 90 "$node_(0) shutdown"
#$ns_ at 90 "$node_(1) shutdown"

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(2) $tcp
$ns_ attach-agent $node_(3) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 60.0 "$ftp start"
#
# 2 connecting to 3 at time 90
#
#$ns_ at 90 "$node_(2) startup"
#$ns_ at 90 "$node_(3) startup"
#set udp_(1) [new Agent/UDP]
#$ns_ attach-agent $node_(2) $udp_(1)
#set null_(1) [new Agent/Null]
#$ns_ attach-agent $node_(3) $null_(1)
#set cbr_(1) [new Application/Traffic/CBR]
#$cbr_(1) set packetSize_ 512
#$cbr_(1) set rate_ 81920
#$cbr_(1) set random_ 1
#$cbr_(1) set maxpkts_ 10000
#$cbr_(1) attach-agent $udp_(1)
#$ns_ connect $udp_(1) $null_(1)
#$ns_ at 90 "$cbr_(1) start"
#$ns_ at 180 "$cbr_(1) stop"
#$ns_ at 180 "$node_(2) shutdown"
#$ns_ at 180 "$node_(3) shutdown"

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
```

```
$ns_ attach-agent $node_(0) $tcp
$ns_ attach-agent $node_(2) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 80.0 "$ftp start"
$ns_ at 90.0 "$ftp stop"

#
# 3 connecting to 4 at time 180
#
#$ns_ at 180 "$node_(3) startup"
#$ns_ at 180 "$node_(4) startup"
#set udp_(2) [new Agent/UDP]
#$ns_ attach-agent $node_(3) $udp_(2)
#set null_(2) [new Agent/Null]
#$ns_ attach-agent $node_(4) $null_(2)
#set cbr_(2) [new Application/Traffic/CBR]
#$cbr_(2) set packetSize_ 512
#$cbr_(2) set rate_ 81920
#$cbr_(2) set random_ 1
#$cbr_(2) set maxpkts_ 10000
#$cbr_(2) attach-agent $udp_(2)
#$ns_ connect $udp_(2) $null_(2)
#$ns_ at 180 "$cbr_(2) start"
#$ns_ at 270 "$cbr_(2) stop"
#$ns_ at 270 "$node_(3) shutdown"
#$ns_ at 270 "$node_(4) shutdown"

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(3) $tcp
$ns_ attach-agent $node_(0) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 65.0 "$ftp start"

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(1) $tcp
$ns_ attach-agent $node_(2) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 65.0 "$ftp start"
```

```
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at 350.0 "$node_($i) reset";
}
$ns_ at 100.0 "stop"
$ns_ at 100.0 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
   global ns_ tracefd
   $ns_ flush-trace
   close $tracefd
exec nam -r 5.0m wirelesstest2.nam &
}

puts "Starting Simulation... with $val(rp)"
$ns_ run
```

- At any time of simulation it can read the value of any variable of the network object write it to the screen or put it into a file. For example to print out on the screen the congestion window size of TCP session:

  » $ns at time “puts [$tcp set cwnd_ ]”

- To Run the simulation type on command prompt in terminal

  ```
  $ ns wifitest.tcl
  $ nam out.nam
  ```

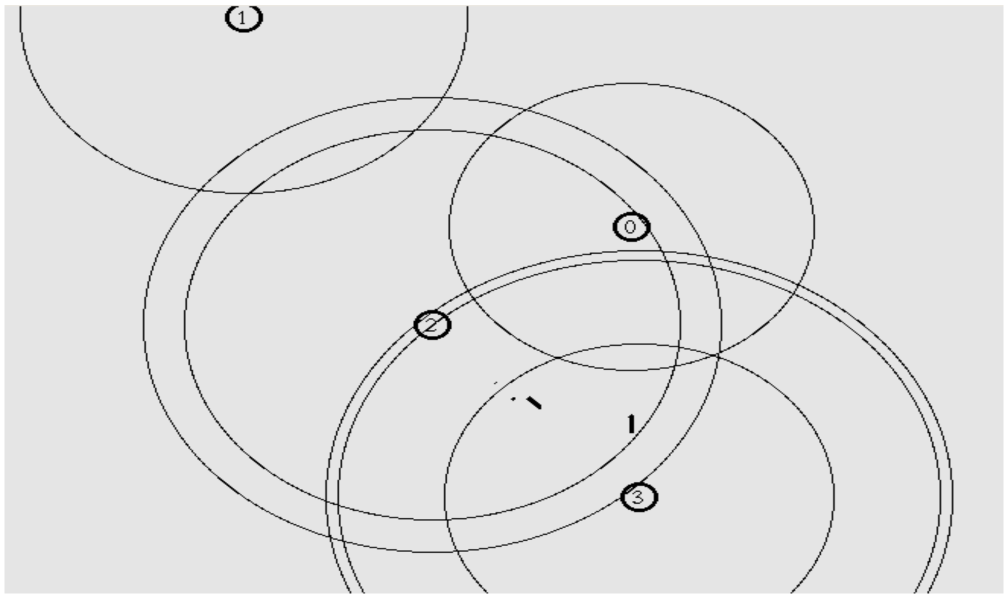


**Figure 2.8: nam output of the wirelesstest2.tcl file**

Running the above script generates a NAM trace file that is going to be used as an input to NAM shown in **Figure 2.8** and a trace file called "mywireless2.tr" that will be used for our simulation analysis. Following shows the trace format and example trace data from “mywireless2.tr".

**# Wirelesstest2.tr output of trace file for wirelesstest2.tcl**

```
M 10.00000 0 (250.16, 320.11, 0.00), (412.80, 392.70), 11.50
M 20.00000 1 (512.51, 475.76, 0.00), (171.60, 556.40), 15.00
M 30.00000 2 (501.04, 569.73, 0.00), (274.34, 299.24), 6.73
M 40.00000 3 (332.33, 81.20, 0.00), (662.75, 470.92), 3.34
s 40.000000000 _0_ AGT  --- 0 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 40.000000000 _0_ RTR  --- 0 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 40.000000000 _0_ RTR  --- 0 AODV 48 [0 0 0 0] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]]
(REQUEST)
s 40.000075000 _0_ MAC  --- 0 AODV 106 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]]
(REQUEST)
r 40.000923444 _2_ MAC  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]]
(REQUEST)
r 40.000923817 _1_ MAC  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]]
(REQUEST)
r 40.000948444 _2_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]]
(REQUEST)
r 40.000948817 _1_ RTR  --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]]
(REQUEST)
s 40.000948817 _1_ RTR  --- 0 AODV 44 [0 0 0 0] ------- [1:255 0:255 30 0] [0x4 1 [1 4] 10.000000]
(REPLY)
s 40.001243817 _1_ MAC  --- 0 ARP 86 [0 ffffffff 1 806] ------- [REQUEST 1/1 0/0]
r 40.001932613 _2_ MAC  --- 0 ARP 28 [0 ffffffff 1 806] ------- [REQUEST 1/1 0/0]
r 40.001932634 _0_ MAC  --- 0 ARP 28 [0 ffffffff 1 806] ------- [REQUEST 1/1 0/0]
```

**# awk file to find throughput of the wireless network.**

```
# ============================throughput.awk =======================
BEGIN {
recvdSize = 0
startTime = 1e6
stopTime = 0
}
{
# Trace line format: normal
if ($2 != "-t") {
event = $1
time = $2
if (event == "+" || event == "-") node_id = $3
if (event == "r" || event == "d") node_id = $4
flow_id = $8
pkt_id = $12
```

```
        pkt_size = $6
        flow_t = $5
        level = "AGT"
        }
        # Trace line format: new
        if ($2 == "-t") {
        event = $1
        time = $3
        node_id = $5
        flow_id = $39
        pkt_id = $41
        pkt_size = $37
        flow_t = $45
        level = $19
        }
        # Store start time
        if (level == "AGT" && (event == "+" || event == "s") && pkt_size >= 512) {
        if (time < startTime) {
        startTime = time
        }
        }
        # Update total received packets' size and store packets arrival time
        if (level == "AGT" && event == "r" && pkt_size >= 512) {
        if (time > stopTime) {
        stopTime = time
        }
        # Rip off the header
        hdr_size = pkt_size % 512
        pkt_size -= hdr_size
        # Store received packet's size
        recvdSize += pkt_size
        }
        }
        END {
        printf("Average Throughput[kbps] = %.2f\t\t
StartTime=%.2f\tStopTime=%.2f\n",(recvdSize/(stopTime-startTime))*(8/1000),startTime,stopTime)
        }
```

**# output after applying awk file to find throughput of the wireless network.**

the command to execute the awk file is

```
1) awk -f throughput.awk wirelesstestlink.tr
```

it gives the o/p as

```
Average Throughput[kbps] = 3573.98          StartTime=67.13      StopTime=707.96
```

```
2)  awk -f throughput.awk wirelesstestlink1.tr
Average Throughput[kbps] = 13156.03         StartTime=36.83      StopTime=585.72
```

## Summary:

In this chapter the study of wired and wireless architecture is discussed and the various types of wireless ad hoc networks. The characteristic, properties and application of wireless ad hoc network is explained. The survey of research work done on the wired and wireless network is also discussed. The overview of the different simulation tools are explained to do the simulation of wired and wireless networks. The detail simulation of the wired and wireless network using network simulator NS-2 is explained with the corresponding NS-2 code and the result derived to evaluate the performance of the network.