## Chapter 3

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Network Overview</td>
<td>67</td>
</tr>
<tr>
<td>3.2</td>
<td>Performance Evaluation</td>
<td>68</td>
</tr>
<tr>
<td>3.3</td>
<td>Features of Network Simulator</td>
<td>69</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Links</td>
<td>70</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Network Nodes</td>
<td>72</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Routing</td>
<td>73</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Routing Protocol Used</td>
<td>73</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Queues</td>
<td>74</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Delay</td>
<td>74</td>
</tr>
<tr>
<td>3.3.7</td>
<td>Agents</td>
<td>74</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Timer</td>
<td>75</td>
</tr>
<tr>
<td>3.3.9</td>
<td>Error Models</td>
<td>75</td>
</tr>
<tr>
<td>3.3.10</td>
<td>Output</td>
<td>75</td>
</tr>
<tr>
<td>3.3.11</td>
<td>Radio Propagation Models</td>
<td>80</td>
</tr>
<tr>
<td>3.3.12</td>
<td>Energy Model</td>
<td>80</td>
</tr>
<tr>
<td>3.3.13</td>
<td>Packet Traversal Details From Layer 3 to Layer 1</td>
<td>80</td>
</tr>
<tr>
<td>3.4</td>
<td>Basic Steps Involved in NS2</td>
<td>81</td>
</tr>
<tr>
<td>3.5</td>
<td>Simulation Scenario</td>
<td>81</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary</td>
<td>82</td>
</tr>
</tbody>
</table>
This chapter presents the features of the Network Simulator so as to provide a simulation scenario which offers a realistic view of a wireless sensor network. The network simulator under study is highlighted along with the performance metrics to be evaluated. The various features of the network simulator to form a network topology, like links and nodes, are presented. The packet format, routing protocols and details about queue, delay, timer, Radio propagation model, Energy model and Error models to depict packet loss are also provided. To analyze the output, the details given in the trace file are elaborated and the simulation scenario is depicted.
In this work, a Channel Adaptive Medium Access Control protocol is designed, which uses multiple techniques to reduce the energy consumption, control overhead and latency, so as to improve the overall system performance. It also considers the effects of link quality, channel quality and energy resources of each node. A Traffic Aware Dynamic Power Management is proposed to enhance the energy efficiency, by reducing the duty cycle of operation. A Load Prediction algorithm with an Adaptive Threshold Adjustment scheme is also designed to prevent the buffer overflow and thus to achieve fairness. Performance requirements of this protocol are validated. For this, Network Simulator NS2 is used. This Channel Adaptive MAC algorithm is evaluated with the existing algorithms to prove its performance. The goal is to increase the energy efficiency in a wireless sensor network, while achieving a high level of stability and scalability.

3.1 NETWORK OVERVIEW

The protocol design in this work, assumes a big wireless sensor network having a large number of sensor nodes, with limited storage, communication and processing capabilities. The nodes are configured in an ad hoc, self-organized and self-managed wireless network fashion. Data generated by the sensors is processed and communicated in a store and forward manner. The applications supported by the network are assumed to swing between long idle periods, during which no events occur, and bursty active periods during which data flows towards the base station, through packet exchange among the peer sensor nodes. The applications are also assumed to tolerate a large latency for an extended network life.

To achieve significant energy savings, this protocol exploits the bursty profile of sensor applications. This is to establish duty cycle operation on nodes in a multi hop network. During the long periods of time when no sensing occurs, the nodes alternate periodically between listening and sleep states. When its radio is
turned OFF, each node sets up a wake up timer for a wake up time and sleeps for a
certain amount of time. Node becomes active on expiration of the timer. This sleep
mechanism reduces the energy consumption. But here nodes do not follow a
periodic sleep mechanism. A Traffic Aware Dynamic Power Management scheme
is adopted here with the aim to increase the energy efficiency and to decrease the
latency, incurring in some existing algorithms. Moreover to increase the energy
efficiency further, the link and channel characteristics are also considered. Most
existing schemes sacrifice fairness, but here in this work, the Load Prediction
algorithm is aimed to prevent this.

3.2 PERFORMANCE EVALUATION

Many performance issues like Throughput, Delay, Energy consumption,
Delivery ratio and so on, critically influence the performance of wireless sensor
networks. So this algorithm is to be validated for these parameters. The algorithm
proposed in this work is evaluated and compared with the existing protocols, by
simulation using the Network Simulator NS2, wherein the real nature of a wireless
sensor environment is created. The reason for selecting NS2 is that, this Network
Simulator uses two languages to enable the simulator to do two tasks which are
discussed below. NS version2 is a discrete event-driven and object-oriented
network simulator. A system programming language is required in detailed
simulations of protocols to efficiently manipulate bytes, packet headers and also to
implement algorithms that run over large data sets. For such tasks, run time speed
is more important compared to turn around time for simulation run, finding bugs
and so on, which happens to be less important. Slightly varying parameters or
configurations, or quickly exploring a large number of scenarios is involved in
detailed network research. For this iteration time is very important. Configuration
runs once at the beginning of the simulation, and so run time of this part of the task
is less important. Both these requirements are met by Network Simulator NS, with two languages, C++ and object oriented extension of Tool Command Language (OTcl). The code is implemented by C++ while OTcl configures the system. OTcl is an interpreted programming language. C++ is suitable for detailed protocol implementation because it is fast to run but slower to change. OTcl is ideal for simulation configuration since it runs slower but can be changed very quickly and interactively.

3.3 FEATURES OF NETWORK SIMULATOR

Network Simulator uses two languages to do two tasks. Front end is OTcl for simulation configuration and C++ is for detailed protocol implementation. Links that connect nodes in the network are OTcl objects that assemble delay, queuing and loss modules. In the starting phase, a new simulator object is created in Tcl which is then followed by the initialization procedure.

![OTcl linkage](image)

**Figure 3.1: OTcl linkage**
Chapter 3

This procedure includes initializing the packet format, creating a scheduler and creation of a null agent as a discard sink. Field offsets are set up within packets in the packet format initialization, to be used during the entire simulation. The simulation is run by the scheduler in an event driven manner. The scheduler selects the next earliest event, executes it to completion, and returns to execute the next event. Unit of time used by scheduler is seconds. The simulator is single-threaded, and only one event is in execution at any given time. If execution of more than one event is to be scheduled at the same time, their execution is performed on the first scheduled – first dispatched manner. The methods used by the simulator class to set up the simulation falls into three categories as methods for creating and managing the topology (which in turn consists of managing the nodes) and managing the links, methods to perform tracing, and helper functions to deal with the scheduler.

3.3.1 Links

Links are used to connect the nodes and complete the topology of a given wired network. A link between two nodes with the specified bandwidth and delay characteristics is created by the Link class. Figure 3.2 shows the components, of a duplex link, between two nodes, n₀ and n₁. The trace elements are tracked by the following instance variables:

\texttt{enqT}_  Reference to the element that traces packets entering the queue.
\texttt{deqT}_  Reference to the element that traces packets leaving the queue.
\texttt{drpT}_  Reference to the element that traces packets dropped from the queue.
\texttt{rcvT}_  Reference to the element that traces packets received by the next node.
Figure 3.2: Duplex link structure

Figure 3.3: Internal structure of nodes

Figure 3.3 shows the internal structure of two nodes \( n_0 \) and \( n_1 \).

Figure 3.4 shows the packet header format.
3.3.2 Network Nodes

All the network nodes have the following components:

- an address or id, monotonically increasing across the simulation namespace by 1 (from initial value 0) as nodes are created,
- a list of neighboring nodes,
- a set of agents,
- a node type identifier, and
- a routing module

Node configuration involves defining the different node characteristics before creating them. Node characteristics comprise of the type of addressing structure used in the simulation, defining the network components for mobile nodes, the Link layer type, MAC type, the interface queue type, the interface queue length, the antenna type, the propagation type, the channel type, turning ON or OFF the trace options at Agent/Router/MAC levels, selecting the type of ad hoc
routing protocol for wireless nodes or defining their energy model. When a node receives a packet, it examines the fields in the packet, usually its destination address, and during some instances, its source address. Then the values to an outgoing interface object that is the next downstream recipient of this packet is mapped by it.

### 3.3.3 Routing

Three functional blocks present in every routing implementation in a Network Simulator are as follows.

- **Routing agent** does the routing packet exchange with the neighbors.
- **Route logic** uses the information gathered by routing agents (or the global topology database in the case of static routing) to perform the actual route computation.
- **Classifiers** present in a node do the address and port classification. The computed routing table is used to perform packet forwarding.

### 3.3.4 Routing Protocol Used

The routing protocol used in simulation is Ad hoc On Demand Distance Vector (AODV). AODV is a combination of both Dynamic Source Routing (DSR) and Destination Sequenced Distance Vector (DSDV) protocols. It not only uses the basic route-discovery and route-maintenance of DSR but also the hop-by-hop routing, sequence numbers and beacons of DSDV. The node in need of a route to a given destination generates a ROUTE REQUEST. The ROUTE REQUEST is now forwarded by the intermediate nodes. During this process, it also creates a reverse route for itself from the destination. When such a request reaches a node with a defined route to the destination, it generates a ROUTE REPLY. This has
information about the number of hops required to reach the destination. All the nodes that participate in forwarding this ROUTE REPLY to the source node, creates a forward route to the destination. This state created from each node from source to destination is not the entire route as in the case of source routing but a hop-by-hop state.

3.3.5 Queues

Queues are the locations where packets are held (or dropped). Packet scheduling involves the decision process which is used to choose which packets should be serviced or dropped. The occupancy of a particular queue is regulated using Buffer management. The present queuing techniques available are drop-tail (FIFO) queuing, Random Early Detection (RED) buffer management, Class Based Queuing (CBQ) (including a priority and round-robin scheduler), and variants of Fair Queuing including Fair Queuing (FQ), Stochastic Fair Queuing (SFQ), and Deficit Round-Robin (DRR). Interface queue used in this work is the Drop tail priority queue.

3.3.6 Delay

Delays represent the time needed for a packet to traverse a link. The possibility of a link failure is captured by a special form of this object (“dynamic link”). The amount of time required for a packet to traverse a link is defined as $s/b + d$ where $s$ is the packet size (as given in its IP header), $b$ is the speed of the link in bits/sec, and $d$ is the link delay in seconds.

3.3.7 Agents

Agents denote the endpoints where the network-layer packets are constructed or consumed, and are used in the implementation of protocols at
different layers. The class Agent has an implementation partly in OTcl and partly in C++.

3.3.8 Timer

Timers may be implemented either in C++ or OTcl. The functions of a Timer Handler comprise of scheduling a timer, rescheduling a timer, canceling a pending timer or returning a timer status.

3.3.9 Error Models

Packet losses are introduced into a simulation using Error models. The link-level errors or loss is simulated by the Error model, by either marking the packet’s error flag or dumping the packet to a drop target. Errors can be generated in simulations, from a simple model such as the packet error rate, or from more complicated statistical and empirical models. The unit of error can be specified in terms of packets, bits, or time-based in order to support a wide variety of models. Corrupted packets are received from Error Model if the drop target exists. Otherwise, Error Model just marks the error flag of the packet’s common header, thus allowing the agents to handle the loss.

3.3.10 Output

Output or trace data during a simulation can be collected in different ways. Commonly trace data is stored in a file to be post-processed and analyzed or generally, trace data is displayed directly during the execution of a simulation. Simulator supports two primary and distinct types of monitoring capabilities. The first one is called trace. Each individual packet as it arrives, departs, or gets dropped at a link or queue, is recorded by it. Like nodes in the network topology, trace objects are configured into a simulation. The second type of object is called
Counts of various quantities such as packet and byte arrivals, departures and so on are recorded by it. A special common header is included in each packet to support traces. Each packet includes a unique identifier, a packet type field which is set by agents when they generate packets, a packet size field in bytes, sent time field which is used to determine the transmission time for packets, and an interface label which is used for computing multicast distribution trees. The trace format is divided into the following fields:

**Event type** This is the first field. It describes the type of event taking place at the node and can be one of the four types:

- **s** send
- **r** receive
- **d** drop
- **f** forward

**General tag** This is the second field. It starts with ":t" which may stand for time or global setting

- **-t** time
- **-t** * (global setting)

**Node property tags** The node properties like node-id, the level at which tracing is being done like agent, router or MAC is depicted by this field. The Node property tags start with a leading ":-N" and are listed as below:

- **-Ni**: node id
- **-Nx**: node’s x-coordinate
- **-Ny**: node’s y-coordinate
- **-Nz**: node’s z-coordinate
- **Ne**: node energy level

- **Ni**: trace level, such as AGT, RTR, MAC

- **Nw**: reason for the event. The different reasons for dropping a packet are given below:

  "END" DROP_END_OF_SIMULATION

  "COL" DROP_MAC_COLLISION

  "DUP" DROP_MAC_DUPLICATE

  "ERR" DROP_MAC_PACKET_ERROR

  "RET" DROP_MAC_RETRY_COUNT_EXCEEDED

  "STA" DROP_MAC_INVALID_STATE

  "BSY" DROP_MAC_BUSY

  "NRTE" DROP_RTR_NO_ROUTE i.e. no route is available.

  "LOOP" DROP_RTR_ROUTE_LOOP i.e. there is a routing loop

  "TTL" DROP_RTR_TTL i.e. TTL has reached zero.

  "TOUT" DROP_RTR_QTIMEOUT i.e. packet has expired.

  "CBK" DROP_RTR_MAC_CALLBACK

  "IFQ" DROP_IFQ_QFULL i.e. no buffer space in IFQ.

  "ARP" DROP_IFQ_ARP_FULL i.e. dropped by ARP

  "OUT" DROP_OUTSIDE_SUBNET i.e. dropped by base stations on receiving routing updates from nodes outside its domain.

**Packet information at IP level** The tags for this field begin with a leading "-I" and are listed along with their explanations as follows:
-Is: source address. source port number

-Id: destination address. destination port number

-It: packet type

-Ii: packet size

-If: flow id

-Ii: unique id

-Iv: time to live (ttl) value

Next hop info This field provides next hop info and the tag starts with a leading "-H".

-Hs: id for this node

-Hd: id for next hop towards the destination.

Packet info at MAC level MAC layer information is given in this field and starts with a leading "-M" as shown below:

-Ma: duration

-Md: ethernet address of destination

-Ms: ethernet address of source

-Mt: ethernet type

Packet info at "Application level" The type of application like Address Resolution Protocol (ARP), Transmission Control Protocol (TCP), the type of ad hoc routing protocol like DSDV, DSR, AODV etc being traced is given in the packet information at the application level. This field consists of a leading "-P" and list of tags for different applications is detailed below:

-P arp Address Resolution Protocol.
ARP details are given by the following tags:

- **Po**: ARP Request/Reply
- **Pm**: source MAC address
- **Ps**: source address
- **Pa**: destination MAC address
- **Pd**: destination address
- **Pdsr**: denotes the ad hoc routing protocol called Dynamic Source Routing.

Information on DSR is given by the following tags:

- **Pn**: number of nodes traversed
- **Pq**: routing request flag
- **Pi**: route request sequence number
- **Pp**: routing reply flag
- **Pl**: reply length
- **Pe**: source of source routing -> destination of the source routing
- **Pw**: error report flag
- **Pm**: number of errors
- **Pc**: whom to report
- **Pb**: link error from link a -> link b

Information about the CBR application is given by the following tags:

- **Pi**: sequence number
- **Pf**: number of times this packet was forwarded
-Po: optimal number of forwards

-P tcp Information about TCP flow is given by the following subtags:

-Ps: sequence number

-Pa: acknowledgement number

-Pf: number of times this packet was forwarded

-Po: optimal number of forwards

3.3.11 Radio Propagation Models

Radio propagation models are implemented in the Network Simulator. The received signal power of each packet can be predicted using these models. There is a receiving threshold at the physical layer of each wireless node. On receiving a packet, if its signal power is below the receiving threshold, it is marked as error and dropped by the MAC layer. Radio propagation model used here is the Two ray ground radio propagation model.

3.3.12 Energy Model

Energy Model is implemented in the Network Simulator as a node attribute. The level of energy in a mobile host is given in the Energy model. The initial value of the level of energy that the node has at the beginning of the simulation is called initialEnergy_. txPower_ and rxPower_ denote the energy used by each packet for transmission/reception. initialEnergy_, txPower_ and rxPower_ are specified in the energy model.

3.3.13 Packet Traversal Details From Layer 3 to Layer 1

Outgoing packets from the Network layer are handed to the Link Layer by the Routing Agent. Link Layer has ARP module which finds the hardware address
of the next hop of packets. Packets are then handed to the Interface Queue which has a maximum queue limit that can be maintained at the Queue. Packets are now handed to the MAC layer and then to the Network Interface which stamps the packet with the meta data details like transmission power, wavelength and so on. This is used by the propagation model in the Network Interface to check whether the packet has the minimum power needed to be received or not.

3.4 BASIC STEPS INVOLVED IN NS2

- Create a new simulator object
- Turn on tracing
- Create the Physical layer and the Network layer
- Create the Link and the Queue
- Define the Routing protocol
- Create the Transport connections
- Create the traffic
- Insert errors to simulate an error prone scenario

3.5 SIMULATION SCENARIO

To create a realistic depiction of a wireless sensor network, a simulation study is performed using Network Simulator NS2. Simulation network consists of many sensor nodes distributed in a grid pattern of 1000×1000 m². Each node is equipped with a radio transceiver capable of transmitting a signal over a distance of 250 m on a 2 Mb/s wireless channel. All applications are run on User Datagram Protocol (UDP). The simulated traffic is of Constant Bit Rate (CBR). The sink is
assumed to be 250m away from the area. Initial node energy is set as 2.7 Joules for the first set of simulations and 4.0 Joules for the second set of simulations.

The Channel Adaptive MAC protocol with Traffic Aware Dynamic Power Management scheme adopts the periodic sleep/listen operations, schedule selection and coordination, schedule synchronization, adaptive listening and access control mechanisms of the SMAC protocol. The mechanisms existing in this protocol are depicted in detail in the succeeding chapters, Chapter 4 and Chapter 5. This protocol is simulated to validate and confirm its superior performance.

### 3.6 SUMMARY

This chapter explains the need to have a realistic depiction of a wireless sensor network, so as to evaluate the proposed Channel Adaptive MAC protocol. The different features of the Network Simulator NS2 are presented. Details of the nodes specified in the Network Simulator are discussed so as to create the wireless sensor network topology. Different aspects of the simulator like the Packet format, Routing protocol, Energy model, Radio propagation model, Error model, Timer and so on are detailed; with a view to show that the simulator can provide a realistic network scenario with packet flows, packet losses and so on. The scenario used for simulation in this research work is illustrated in the last section.