CHAPTER 3

PERFORMANCE ANALYSIS OF AD HOC ON DEMAND DISTANCE VECTOR ROUTING PROTOCOL

Wireless networking research is fundamentally dependent upon simulation. JiST can be used to construct efficient, robust and scalable simulations.

3.1 ARCHITECTURE OF JiST

The benefits of this approach to simulator construction over traditional systems and languages approaches are numerous. Embedding the simulation semantics within the Java language allows us to reuse a large body of work, including the Java language itself, its standard libraries and existing compilers. JiST benefits from the automatic garbage collection, type-safety, reflection and many other properties of the Java language. The overall architecture of JiST is shown in Figure 3.1.

(Source: Lin (2004)

Figure 3.1 The JiST system architecture)
3.2 SWANS DESIGN HIGHLIGHTS

The SWANS software is organized as independent software components that can be composed to form complete wireless network or sensor network simulations, as shown in Figure 3.2. Its capabilities are similar to those of NS2 and GloMoSim, two popular wireless network simulators. There are components that implement different types of applications; networking, routing and media access protocols; radio transmission, reception and noise models; signal propagation and fading models; and node mobility models. Instances of each component type are shown italicized in the figure.

The development of SWANS has been relatively easy. Since JiST inter-entity message creation and delivery is implicit, as well as message garbage collection and typing, the code is compact and intuitive. Components in JiST consume less than half of the code (in uncommented line counts) of comparable components in GloMoSim, which are already smaller than their counterpart implementations in NS2.

![Figure 3.2 The SWANS Architecture](image_url)
Every SWANS component is encapsulated as a JiST entity: it stores its own local state and interacts with the other components via exposed event-based interfaces. SWANS contains components for constructing a node stack, as well as components for a variety of mobility models and field configurations. This pattern simplifies simulation development by reducing the problem to creating relatively small, event-driven components. It also explicitly partitions the simulation state and the degree of inter-dependence between components, unlike the design of NS2 and GloMoSim. It also allows components to be readily interchanged with suitable alternate implementations of the common interfaces and for each simulated node to be independently configured. Finally, it also confines the simulation communication pattern.

It is important to note that, in JiST, communication among entities is very efficient. The design incurs no serialization, copy, or context-switching cost among co-located entities, since the Java objects contained within events are passed along by reference via the simulation time kernel. Simulated network packets are actually a chain of nested objects that mimic the chain of packet headers added by the network stack. Moreover, since the packets are timeless by design, a single broadcasted packet can be safely shared among all the receiving nodes and the very same object sent by an entity on one node will be received at the same entity of another node.

Dynamically created objects such as packets can traverse many different control paths within the simulator and can have highly variable lifetimes. The accounting for when to free unused packets is handled entirely by the garbage collector. This not only simplifies the memory management protocol, but also eliminates a common source of memory leaks that can accumulate over long simulation runs.
The partitioning of node functionality into individual, fine-grained entities provides an additional degree of flexibility for distributed simulations. The entities can be vertically aggregated, as in GloMoSim, which allows communication along a network stack within a node to occur more efficiently. However, the entities can also be horizontally aggregated to allow communication across nodes to occur more efficiently. In JiST, this reconfiguration can happen without any change to the entities themselves. The distribution of entities across physical hosts running the simulation can be changed dynamically in response to simulation communication patterns and it does not need to be homogeneous.

3.3 PERFORMANCE

The performance of JiST was compared with the two most popular ad hoc network simulators such as NS2 and GloMoSim. The UDP-based beaconing Node Discovery Protocol (NDP) was used for the performance analysis. NDP was chosen as it was an integral part of most of the ad hoc network protocols and applications. Identical scenario was developed in each of the simulation platform to facilitate the study.

The throughput results were plotted both on log-log and linear scales as shown in Figure 3.3. As expected, the simulation times are quadratic functions of n, the number of nodes, when using the naïve signal propagation algorithm. Even without node mobility, NS2 is highly inefficient. SWANS outperforms GloMoSim by a factor of 2. SWANS-hier uses the improved hierarchical binning algorithm to perform signal propagation instead of scanning through all the radios. As expected, SWANS-hier scales linearly with the number of nodes. The graph depicts clearly that SWANS significantly outperforms NS2 and GlomoSim in the simulation of NDP.
Figure 3.3. Throughput results

The memory footprint results were plotted as shown in Figure 3.4 on log-log scale. JiST is more efficient than GloMoSim and NS2 by almost an order and two orders of magnitude, respectively. This allows SWANS to simulate much larger networks. The memory overhead of hierarchical binning is asymptotically negligible.

Figure 3.4. Memory Usage
3.4 POISSON DISTRIBUTION

The Poisson distribution can be used for modeling applications that involve inter-arrival time. Multimedia wireless networks involve the transfer of messages which is evidently used in presentation in multiple media. In this scenario, there are messages, not certainly data messages, arriving at each node. The time between the arrivals of two consecutive messages can be regarded as the inter-arrival time. In most of these situations, it is assumed that the packets follow a Poisson distribution. This is supported by a number of technical papers that focus on the characteristics of the packets. Some examples are Traffic Dimensioning for Multimedia Wireless Networks by Ribeiro (2004), Traffic Engineering and QoS Control in Multimedia – enabled Wireless Networks by (Kouchryavy & Giambene 2003). Quality of Service Investigation for Multimedia Transmission over Wireless Local Area Networks by El Fishaway (2004). It has been proved that the other distributions cannot appreciably model the scenario.

Summarizing, Poisson distribution can be used in situations that involve inter-arrival time of events. Multimedia messages are analogous to scenario of this genre and Poisson distribution can be used for modeling.

3.5 SIMULATION DESIGN AND RESULTS

The simulation is developed based on the facts explained in the earlier chapters. The simulation engine is used to model an ad hoc wireless network deploying the Ad hoc On-demand Distance Vector routing protocol and communicating multimedia packets among themselves. The simulation uses largely the libraries provided by the simulator. The key issues of any routing protocol such as throughput, latency and control packet overhead play a vital role in analyzing the performance of the protocol.
3.5.1 Simulation Design

The objective is to simulate the communication of multimedia packets in ad hoc network deploying the AODV routing protocol. The packets are Poissonian in nature. Hence, the prominent tasks in simulation are developing classes that accomplish

- Generating Multimedia Packets
- Generating Poisson Variables
- Porting to AODV simulation

3.5.2 Generating Multimedia Packets

Multimedia data are communicated in the ad hoc network using multimedia streams. The multimedia streams are of a fixed size represented in Kilo bytes. A multimedia stream consists of a number of packets called the multimedia packets. Alike multimedia streams, the size of multimedia packets also have a range. In the implementation, multimedia packets are simulated using the class MediaPacket. The description of the class is presented as follows:

Class : MediaPacket, Members : MESSAGE_SIZE, Type: Integer

Methods : getSize() - returns an integer representing the size of the MultimediaPacket.

Constructor : MediaPacket(int messageSize) - creates an instance of the MediaPacket() class with messageSize as the size of the MediaPacket

Multimedia streams are simulated using the class MediaStream. The description of the class is presented as follows:

Class : MediaStream, Members : total Media Length, Type: Integer

Methods : getMediaPacket(int length) - returns a MediaPacket with length as the size of the MediaPacket()
Constructor : MediaStream(int totalMediaLength) – creates an instance of the MediaStream() class with totalMediaLength as the size of the Media stream

Size of the media stream is assigned as 128KB. Multimedia packets are generated until the sum of all media packets equals the size of the media stream. The size of the media packets is constrained between 10 and 100.

3.5.3 Generating Poisson Variable

In a typical multimedia wireless network, the data packets, that is media packets, are assumed to follow Poisson distribution. Poisson variables are generated and supplied as the size of the media packets. The Poisson variables are generated using the class GeneratePoisson. The description of the class is presented as follows:

Class : GeneratePoisson, Members : poissonVarList, Type: Linked List

Methods

- poissonList(int num) – returns a linked list of Poisson variables. The argument num specifies the number of Poisson variables required.

- Poisson Val(double x, double lambda) – returns a double value representing the Poisson variable for a particular value of x and with mean as lambda. The computation is governed by the Equation (3.1).

\[ f(k, \lambda) = \frac{e^{-\lambda} \lambda^k}{k!}, \quad (3.1) \]

- Factorial(int x) – returns an integer representing the factorial value of x.
- *Stabilize*(double *val*) – returns an integer which is the representative (falling between 10 and 100) of the double value *val*.

Data flow diagrams can be used to depict the flow of data in a system or a particular module of the system. The generation of media packets is shown in the Data Flow Diagram in Figure 3.5.

![Data Flow Diagram](image)

**Figure 3.5 Data Flow Diagram for media packet generation**

### 3.5.4 Porting To Aodv Simulation

Primarily, the simulation calculates the *number of messages* from the arguments such as send rate, number of nodes, and duration. This parameter corresponds to the number of media packets that can be sent. This
is the argument that corresponds to the size of the linked list containing Poisson variables. The variables are used to generate the media packets.

Secondarily, transport layer UDP messages are created which contain the media packet attached to them as data. The User Datagram Protocol is used at this stage.

Finally, network layer messages are created through which the UDP messages are transmitted to the destination. This is depicted in Figure 3.6 as a Data Flow Diagram.

![Data Flow Diagram](image)

**Figure 3.6 Data Flow Diagram for Porting to AODV simulation**
3.5.5 Simulation Results

The prominent parameters that can be used to analyze the performance of a routing protocol are throughput, latency, and control packet overhead. The simulation is run with varying values of number of nodes, mobility, and arrangement. The desired parameters are tabulated and graphs are plotted. The simulation results can be categorized as follows:

3.5.5.1 Throughput results

Throughput can be defined as the ratio size of messages sent to the time needed for sending those messages. In the following analysis, it is expressed in Kilo bytes per second – kbps. The time needed for sending all the messages is recorded and throughput is calculated. Two distinct scenarios are considered in throughput analysis, one with nodes arranged in rectangular grids and the other with random arrangement. The result of throughput rates are shown in Table 3.1.

Table 3.1 Throughput Results

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Throughput in kbps for random arrangement</th>
<th>Throughput in kbps for rectangular grid arrangement</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>455.52</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>272.92</td>
<td>315.27</td>
</tr>
<tr>
<td>15</td>
<td>178.03</td>
<td>199.69</td>
</tr>
<tr>
<td>20</td>
<td>134.31</td>
<td>149.01</td>
</tr>
<tr>
<td>25</td>
<td>84.49</td>
<td>98.69</td>
</tr>
<tr>
<td>30</td>
<td>68.82</td>
<td>68.82</td>
</tr>
<tr>
<td>35</td>
<td>56.51</td>
<td>41.80</td>
</tr>
</tbody>
</table>
The results indicate that the throughput is higher for nodes arranged in rectangular grids when the number of nodes is \textit{less than thirty}. With increasing number of nodes, the throughput gradually drops down. Hence, nodes can be arranged in rectangular grids to achieve greater throughput, provided the number of nodes is not higher than thirty. This is shown in the graph in Figure 3.7.

![Throughput in Kbps](image)

**Figure 3.7 Throughput Results**

### 3.5.5.2 Latency results

Latency refers to the delay experienced in sending a packet from the source to the destination. To calculate the delay, the time needed to send all the packets was recorded. Graphs were plotted as shown in Figure 3.8. It clearly shows that the delay incurred in sending media packets increases with the increase in the number of nodes.
3.5.5.3 Control packet overhead

Control packet overhead is realized by plotting the number of control packets that is the protocol messages to the data messages. From this analysis, the number of control messages needed to send a particular number of data messages can be visualized. This is shown in Figure 3.9.
3.5.5.4 Effect of mobility

Mobility, a prominent issue in ad hoc networks, has an effect on the performance of the protocol. Consider a situation in which the route discovery process is completed. Suppose a node on the chosen route moves away, RouteError messages are transmitted to the source and other nodes. Subsequently, route discovery process is re-initiated. This involves in re-iteration for route establishment between the source and the destination. Hence, additional control packets and time are needed to establish the route again. The effect of mobility is analyzed by recording the number of RouteError RERR messages generated. The values were obtained for two kinds of mobility, one with mobility static and the other in which the nodes walk within the specified radius. The results were plotted as shown in Figure 3.10.

Figure 3.10. Effect of Mobility
In summary, the performance of the AODV protocol in sending multimedia packets was analyzed with various situations and it was observed that mobility, number of nodes, and their arrangement had an effect on the performance of the protocol.

### 3.6 INFERENCES FROM THE SIMULATION RESULTS

The following inferences were made from the simulation results:

1. To ensure higher throughput, the nodes should be arranged in rectangular grids, provided the number of nodes is less than twenty five, approximately.
2. Delay experienced in sending the data messages increases with increasing number of nodes.
3. Alike delay, the control packet overhead also increases with increasing number of nodes.
4. Mobility has an impact on the performance of the protocol. Increased mobility increases the control packet overhead and delay.

Thus the multimedia packets were simulated and a Poisson variable generation engine was used to specify the sizes of the media packets. The generated packets were ported to AODV simulation. The simulation results were tabulated and corresponding graphs were also drawn.

In conclusion, certain optimization can be done to the protocol to reduce the increases of delay and control packet overhead with increasing number of nodes. In most cases, AODV works well with dynamic link conditions, mobility and imposes low memory overhead in achieving on-demand routing for ad hoc wireless networks.