Islamic University of Technology

CSE 4840

Internetworking Protocols Lab

Lab 3

Mobility Model and Routing Protocols in MANET

Name: Aashnan Rahman

ID: 190041204

Section 2B

April 1, 2024

Contents

Mobility Model and Routing Protocols in MANET

1 MANET

MANET stands for Mobile Ad-hoc Network. It is a type of wireless network consisting of mobile devices that communicate with each other without the need for a fixed infrastructure or centralized administration. In MANETs, devices such as laptops, smartphones, tablets, sensors, and vehicles form a dynamic and self-configuring network, enabling communication in scenarios where traditional wired or infrastructure-based networks are impractical or unavailable.

1.1 Features

The features of MANET are as follows -

- **Decentralized:** No fixed routers or access points. Devices themselves participate in routing and forwarding packets for each other.
- **Dynamic Topology:** Nodes (devices) in a MANET can move around freely, causing the network topology (connections between devices) to change constantly.
- **Self-Configuring:** Devices automatically discover each other and establish routes to communicate. Routing protocols are essential for this dynamic configuration.
- **Limited Range:** Each device has a limited transmission range. Nodes rely on multi-hop communication, where packets are relayed by other nodes to reach their destination if it's beyond the direct range.

1.2 Challenges

Mobile Ad-hoc Networks encounter numerous obstacles and challenges such as -

- **Routing:** Maintaining efficient routing protocols to find paths in a constantly changing network topology is crucial.
- **Limited Resources:** Mobile devices often have limited battery power, processing capabilities, and storage space. Designing efficient protocols that minimize resource consumption is important.
- **Scalability:** As the number of devices in the network increases, routing and maintaining connectivity can become more complex.

2 Classification of MANET

MANET can be classified into the following categories -

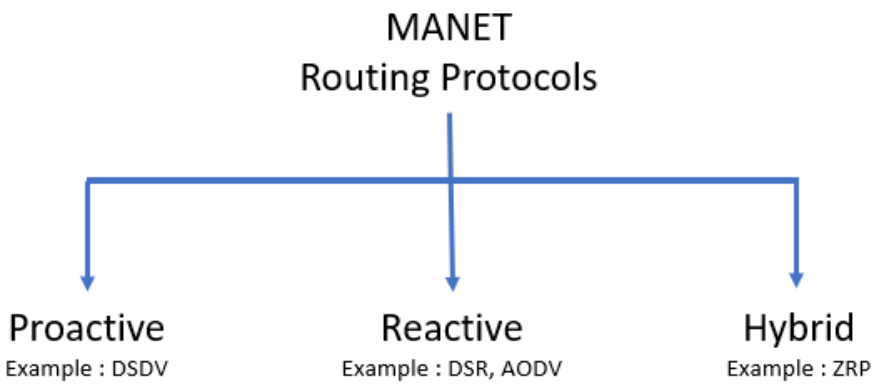

Figure 1: MANET Classification

2.1 Proactive :

Proactive MANET routing protocols establish and maintain routes between nodes in advance, regardless of whether data needs to be transmitted. These protocols periodically exchange routing information between neighboring nodes to ensure that up-to-date routing tables are maintained throughout the network.
Example : DSDV, OSLR

2.1.1 Destination-Sequenced Distance Vector Routing (DSDV)

DSDV (Destination-Sequenced Distance Vector) is a proactive routing protocol used in Mobile Ad-hoc Networks (MANETs). Unlike reactive protocols that establish routes only when needed, DSDV maintains a routing table containing information about routes to destinations, even if they are not actively being used. This table is periodically updated and distributed to neighboring nodes, ensuring that all nodes have consistent routing information. DSDV uses sequence numbers to tag routing updates, allowing nodes to differentiate between old and new routing information. This helps in preventing routing loops and ensuring the freshness of route updates. Overall, DSDV is suitable for relatively stable networks where frequent changes in topology are not expected, providing a simple and efficient routing solution for MANETs.

2.1.2 Optimized Link State Routing (OSLR)

OSLR (Optimized Link State Routing) is a proactive routing protocol designed for Mobile Ad-hoc Networks (MANETs). It optimizes routing by using Multipoint Relays (MPRs) to

reduce flooding overhead. This allows for efficient communication in dynamic networks with limited resources. OSLR is standardized by the IETF and supports Quality of Service (QoS) parameters, making it widely adopted and interoperable.

2.2 Reactive :

Reactive MANET routing protocols, also known as on-demand routing protocols, establish routes between nodes only when needed, rather than continuously maintaining routes proactively. These protocols utilize route discovery mechanisms to find a path to the destination node upon receiving a data packet.
Example : DSR, AODV

2.2.1 Dynamic Source Routing (DSR)

DSR (Dynamic Source Routing) is a reactive routing protocol widely used in Mobile Ad-hoc Networks (MANETs). Unlike traditional routing protocols that maintain routing tables, DSR relies on source routing, where each data packet carries the complete path to its destination. When a node wants to send data to a destination, it initiates a route discovery process by broadcasting a route request (RREQ) packet. Nodes that receive the RREQ and have a fresh route to the destination or know a route to the destination will send back a route reply (RREP) packet containing the complete route. The source node caches the received route, and subsequent data packets are sent along this route. DSR is highly flexible and adaptive to network changes, as routes are only discovered when needed, reducing overhead and adapting to dynamic network conditions. However, the reliance on source routing can result in larger packet sizes and increased overhead.

2.2.2 Ad hoc On-Demand Distance Vector (AODV)

AODV (Ad hoc On-Demand Distance Vector) is a reactive routing protocol commonly used in Mobile Ad-hoc Networks (MANETs). Unlike proactive protocols that maintain routing information for all nodes in the network, AODV establishes routes only when needed. When a node wants to send data to a destination for which it does not have a route, it initiates a route discovery process. During route discovery, the node broadcasts a route request (RREQ) packet, which is forwarded by other nodes towards the destination. When the RREQ reaches the destination or a node with a fresh route to the destination, a route reply (RREP) packet is sent back to the source node, establishing a route. AODV uses sequence numbers to ensure the freshness of routing information and prevent routing loops. Overall, AODV is efficient for dynamic and mobile networks, as routes are established only when needed, reducing overhead and conserving network resources.

2.3 Hybrid :

Hybrid MANET routing protocols combine aspects of both proactive and reactive routing approaches to optimize routing performance in dynamic network environments. These protocols maintain a subset of routes proactively while establishing additional routes reactively as needed. This hybrid approach aims to strike a balance between the low latency of reactive protocols and the reduced overhead of proactive protocols.
Example : ZRP

2.3.1 Zone Routing Protocol (ZRP)

ZRP (Zone Routing Protocol) is a hybrid routing protocol designed for Mobile Ad-hoc Networks (MANETs). It divides the network into zones, with each node maintaining routing information for its local zone. Within a zone, proactive routing is used to maintain routes, while reactive routing is employed between zones. This hybrid approach balances the overhead of proactive protocols and the latency of reactive protocols, making ZRP efficient for dynamic networks.

3 Comparison of Routing Protocols

Feature	AODV (Ad hoc On-Demand Distance Vector)	DSR (Dynamic Source Routing)	OSLR (Optimized Link State Routing)	DSDV (Destination-Sequenced Distance Vector)
Type	Reactive	Reactive	Proactive	Proactive
Route Discovery	On-demand	On-demand	Pre-calculated	Pre-calculated
Routing Table	Maintained at each node, stores routes to known destinations	Not maintained at nodes, routes discovered dynamically	Maintained at each node, stores routes to all destinations	Maintained at each node, stores routes to all destinations with sequence ID
Route Maintenance	Routes updated with route error messages or periodic beaconing	Routes maintained by route lifetime timers and repair mechanisms	Routes updated with link-state advertisements	Routes updated with periodic updates and sequence numbers
Overhead	Lower overhead compared to proactive protocols	Potentially higher overhead due to route discovery traffic	Potentially lower overhead compared to DSDV	Higher overhead due to periodic full RT updates
Convergence Time	Faster route discovery for known destinations	Can be slower for initial route discovery	Fast route convergence after topology changes	Fast route convergence after topology changes
Scalability	Potentially better scalability for larger networks	Potentially lower scalability due to route discovery overhead	Potentially better scalability than DSDV	Lower scalability due to full routing table updates
Security	Requires additional security mechanisms	Requires additional security mechanisms	Requires additional security mechanisms	Requires additional security mechanisms

4 Mobility Models

Mobility models are crucial for simulating and analyzing the behavior of Mobile Ad-hoc Networks (MANETs). These models represent how nodes (devices) move within the network, affecting network performance and routing protocols. The explanation of three common mobility models used in MANET simulations are as follows-

4.1 Random Waypoint

The Random Waypoint model simulates node movement by randomly selecting a destination within a defined area and then moving towards that destination at a constant speed. Upon reaching the destination, the node pauses for a certain amount of time before selecting a new random destination and repeating the process. This model is commonly used to represent human or vehicle movement in urban environments and is characterized by periods of both movement and rest.

4.2 Random Walk

The Random Walk (RW) mobility model, inspired by Brownian Motion, simulates a mobile node's movement by randomly selecting a direction and speed within predefined ranges. Movements occur either in constant time intervals or after a fixed distance, with new directions and speeds calculated for each step. This model shares similarities with the Random Waypoint model due to their randomness. The Random Walk can be seen as a specific instance of the Random Waypoint model with no pause time. It's valuable for scenarios where nodes display erratic behavior or are influenced by external factors, like animals in wildlife monitoring.

4.3 Group

The Group Mobility Model represents a scenario where nodes move in groups or clusters, rather than independently. This model simulates situations where nodes tend to move together due to common goals, tasks, or environmental factors.

4.4 Graphical Representation of Mobility Models

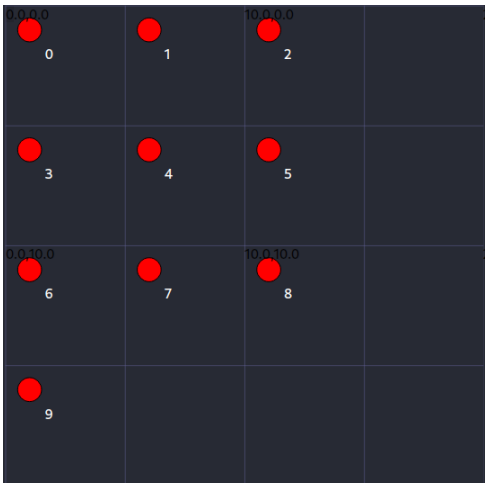

Figure 2: Start

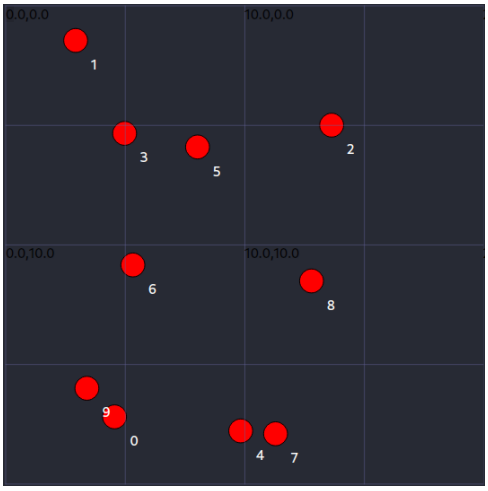

Figure 3: End

Figure 4: Random Walk Mobility Model

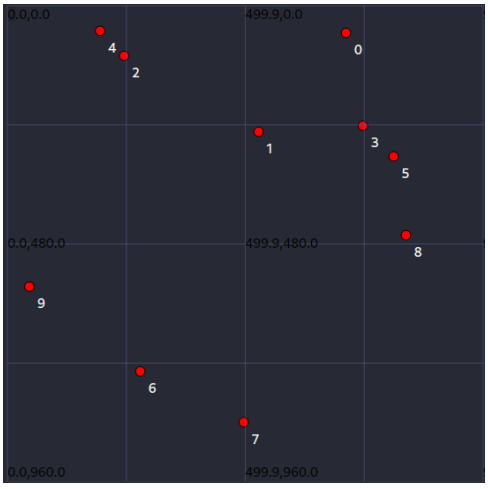

Figure 5: Start

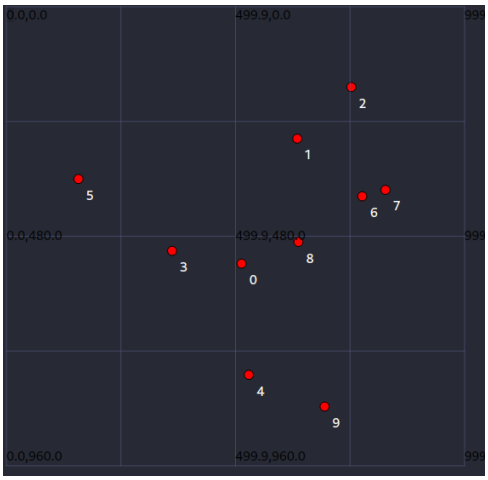

Figure 6: End

Figure 7: Random Waypoint Mobility Model

5 Comparison of Mobility Models

The comparison between Random Walk and Random Waypoint Mobility model has been shown below-

Characteristics	**Random Waypoint Model**	**Random Walk Model**
Proposed By	Johnson and Maltz	Karl Pearson
Key Features	Vmax is maximum speed and Tpause is the stop time upon reaching destination	Specific type of Random Waypoint with Tpause = 0
Node Distribution Method	Uniform	Uniform
Memory Status	Memoryless	Memoryless
Average Speed	Between(0,Vmax)	At interval t, node moves (t) from (0,2π)
Distribution Method	Probability Distribution	Uniform/Gaussian Distribution
Group Mobility Model	Entity Mobility Model	Entity Mobility Model
Temporal Dependency	None	None
Spatial Dependency	None	None
Geographic Restriction	None	None

6 Result Analysis

Comparison of packet receive rates of different MANET routing protocols on different mobility models are as follows -

6.1 Tabular Comparison

Mobility Models	DSDV	OSLR	AODV	DSR
Random Walk	7.69	8.27	8.00	8.04
Random Waypoint	3.73	3.96	7.51	5.40

Table 1: Comparison of packet receive rates

6.2 Graphical Comparison

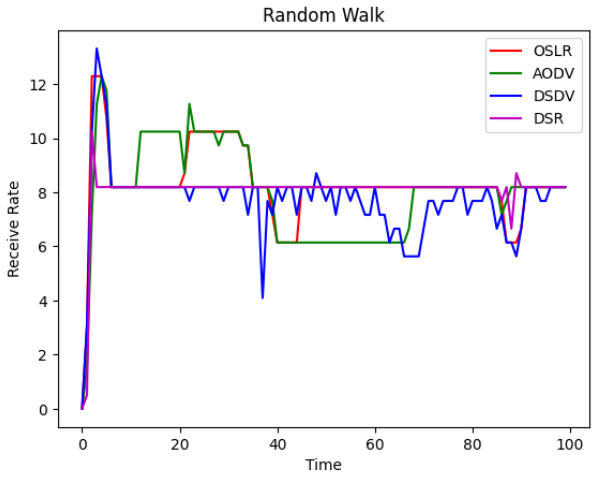

Figure 8: Random Walk

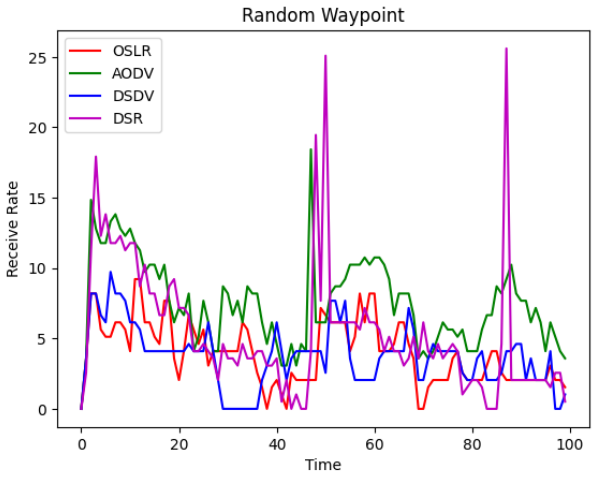

Figure 9: Random Waypoint

Figure 10: Random Walk Mobility Model

6.3 Explanation of Result

- **Randow Waypoint**
  From the graph we can observe that in the Random Waypoint mobility model, reactive MANET routing protocols like DSR and AODV are performs comparatively well due to the dynamic nature of the network. Among them, AODV is slightly favored over DSR because it generally exhibits lower route discovery and maintenance overhead. AODV's efficient route discovery process and periodic route maintenance mechanisms contribute to its suitability for highly dynamic environments like those found in the Random Waypoint mobility model.

- **Random Walk**
  It can be seen that in the random walk mobility model, all routing algorithms yield similar results due to the model's inherent unpredictability and randomness in node movements. This uniformity arises because the routing overhead remains consistent across different protocols, and the random nature of node movements limits optimization opportunities. With little correlation among node movements, there's a lack of distinct performance variation among routing protocols in this scenario. Therefore, network behavior tends to be comparable across different routing algorithms in the random walk mobility model.

7 Code Breakdown

7.1 Routing Protocols

- **OnOffApplication**:
  The OnOffApplication module is used to simulate traffic generated by an application that periodically turns on and off. It is a type of traffic generator commonly used for simulating bursty or intermittent traffic patterns in network simulations.

  The OnOffApplication works by alternating between two states: ON and OFF. During the ON state, the application generates and sends data packets at a specified data rate. During the OFF state, no data packets are generated or sent. This ON-OFF cycle continues for a specified duration or until a stop condition is met.

- **WifiRemoteStationManager** ;
  WifiRemoteStationManager is a module used to manage the configuration and behavior of Wi-Fi stations (STAs) in a simulated Wi-Fi network. It provides control over various aspects of station operation, including transmission parameters, channel access mechanisms, and station characteristics.

  - **Non-Unicast Mode:**
    Configuring the non-unicast mode parameters for stations, which determine how stations handle non-unicast (broadcast and multicast) traffic.
  - **RTS/CST Threshold**
    Setting the threshold value for the Request to Send/Clear to Send (RTS/CTS) mechanism, which determines when stations should use the RTS/CTS exchange to avoid collisions in the transmission of data packets.

7.2 Mobility Models

- **Mobility Helper**:
  MobilityHelper is a helpful class in NS-3 for assigning positions and mobility models to nodes in your network simulations. It simplifies the process of configuring how nodes move within the simulation environment.

  - **GridPositionAllocator:** Places nodes in a grid layout.
  - **ConstantPositionMobilityModel:** Nodes remain stationary throughout the simulation.

7.2.1 Random Walk

```
int main(int argc, char *argv[]) {
  Time::SetResolution(Time::MS);
  NodeContainer nodes;
  nodes.Create(10); // Create 10 nodes

  MobilityHelper mobility;
  mobility.SetPositionAllocator("ns3::GridPositionAllocator", "MinX",
    DoubleValue(1.0), "MinY", DoubleValue(1.0), "DeltaX",
  DoubleValue(5.0),
  "DeltaY",
  DoubleValue(5.0),
  "GridWidth",
  UintegerValue(3),
  "LayoutType",
  StringValue("RowFirst"));
  // Set up the Random Walk 2d mobility model
  mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel", "Bounds",
    RectangleValue(Rectangle(0, 20, 0, 20)),  "Distance", DoubleValue(10),
   "Speed",
    StringValue("ns3::ConstantRandomVariable[Constant=1.0]") );
  mobility.Install(nodes);

  AnimationInterface anim("Random_Waypoint.xml");
  Simulator::Stop(Seconds(100.0));
  Simulator::Run();
  Simulator::Destroy();
  return 0;
}
```

This code creates a container of 10 nodes, sets up a grid-based position allocator for these nodes with specified parameters, and then installs a 2D Random Walk mobility model for each node within a defined rectangular area (Bounds). The nodes move randomly with a constant speed of 1.0 units per time step within the specified bounds, taking steps of 10 units at each movement event.

7.2.2 Random Waypoint

```
int main(int argc, char *argv[]) {
  Time::SetResolution(Time::MS);
  NodeContainer nodes;
  nodes.Create(10); // Create 10 nodes

  MobilityHelper mobility;
      ObjectFactory pos;
      pos.SetTypeId("ns3::RandomRectanglePositionAllocator");
      pos.Set("X", StringValue("ns3::UniformRandomVariable[Min=0.0|Max
    =1000.0]"));
      pos.Set("Y", StringValue("ns3::UniformRandomVariable[Min=0.0|Max
    =1000.0]"));

      std::ostringstream speedConstantRandomVariableStream;
      speedConstantRandomVariableStream << "ns3::ConstantRandomVariable[
    Constant=" << 50 << "]";

      Ptr<PositionAllocator> taPositionAlloc = pos.Create()->GetObject<
    PositionAllocator>();
      mobility.SetMobilityModel("ns3::RandomWaypointMobilityModel",
                            "Speed",
                            StringValue(speedConstantRandomVariableStream.str
    ()),
                            "Pause",
                            StringValue("ns3::ConstantRandomVariable[Constant
    =2.0]"),
                            "PositionAllocator",
                            PointerValue(taPositionAlloc));
      mobility.SetPositionAllocator(taPositionAlloc);
      mobility.Install(nodes);

  AnimationInterface anim("Random_Waypoint.xml");
  Simulator::Stop(Seconds(100.0));
  Simulator::Run();
  Simulator::Destroy();
  return 0;
}
```

This code creates a container of 10 nodes and sets up a Random Waypoint mobility model for each node. The mobility model specifies that nodes will move within a rectangular area with randomly assigned positions. The speed of movement is constant at 50 units per time step, and nodes pause for 2 seconds between movements. Finally, the mobility model is installed on the nodes, allowing them to move according to the specified parameters.

7.3 Device

- **WifiMacHelper :**
  WifiMacHelper is a crucial class that assists in configuring and installing Wifi MAC (Medium Access Control) objects on nodes.

  a) **AdhocWifiMac :** Represents a node in an ad-hoc network (without a central AP).

- **YansWifiPhyHelper :**
  YansWifiPhyHelper plays a vital role in configuring the physical (PHY) layer of the Wifi channel. It provides a simplified interface for creating and managing the PHY objects based on the popular Yans Wifi model.

7.4 Solution

a) **ReceivePacket**

```
void
RoutingExperiment::ReceivePacket(Ptr<Socket> socket)
{
    Ptr<Packet> packet;
    Address senderAddress;
    while ((packet = socket->RecvFrom(senderAddress)))
    {
        bytesTotal += packet->GetSize();
        packetsReceived += 1;
        NS_LOG_UNCOND(PrintReceivedPacket(socket, packet,
    senderAddress));
    }
}
```

The 'RoutingExperiment::ReceivePacket' function serves as a callback handler for receiving packets through a socket. Upon invocation, it retrieves packets from the socket's receive buffer one by one, updating statistics such as total bytes received and the count of received packets. For each received packet, it logs information including the socket, packet contents, and sender's address for debugging or analysis purposes. This function facilitates packet reception and data collection in network experiments or simulations.

b) **CheckThroughput**

```
void
RoutingExperiment::CheckThroughput()
{
    double kbs = (bytesTotal * 8.0) / 1000;
    bytesTotal = 0;

    std::ofstream out(m_CSVfileName, std::ios::app);

    out << (Simulator::Now()).GetSeconds() << "," << kbs << "," <<
    packetsReceived << ","
        << m_nSinks << "," << m_protocolName << "," << m_txp << "" <<
    std::endl;


    out.close();
    packetsReceived = 0;
```

```
    Simulator::Schedule(Seconds(1.0), &RoutingExperiment::
    CheckThroughput, this);
}
```

The 'RoutingExperiment::CheckThroughput' function calculates and logs network throughput periodically. It computes throughput in kilobits per second based on the total bytes received, resets the byte counter, and appends throughput data to a CSV file along with other simulation parameters. It then schedules the next throughput check to occur after one second. This function effectively monitors and records network throughput during the simulation.

c) **Routing Protocol and Mobility Model**

```
std::string mobility_type;
    //mobility_type = "Random Waypoint";
    // mobility_type = "Group";
    mobility_type = "Random Walk";
    if(mobility_type == "Random Waypoint") {
        mobilityAdhoc.SetMobilityModel("ns3::
    RandomWaypointMobilityModel",
                                    "Speed",
                                    StringValue(ssSpeed.str()),
                                    "Pause",
                                    StringValue(ssPause.str()),
                                    "PositionAllocator",
                                    PointerValue(taPositionAlloc));
    } else if (mobility_type == "Random Walk")
    {
        mobilityAdhoc.SetMobilityModel("ns3::RandomWalk2dMobilityModel
    ",
                                       "Speed",
                                       StringValue("ns3::
    UniformRandomVariable[Min=0.0|Max=20.0]"),
                                       "Bounds",
                                       RectangleValue(Rectangle(0,
    300, 0, 1500))
                                       );
    } else if (mobility_type == "Group")
    {
        // Add group mobility code here
    } else {
        NS_FATAL_ERROR("No such mobility model:" << mobility_type);
    }


    mobilityAdhoc.SetPositionAllocator(taPositionAlloc);
    mobilityAdhoc.Install(adhocNodes);
```

```
    streamIndex += mobilityAdhoc.AssignStreams(adhocNodes, streamIndex
    );

    AodvHelper aodv;
    OlsrHelper olsr;
    DsdvHelper dsdv;
    DsrHelper dsr;
    DsrMainHelper dsrMain;
    Ipv4ListRoutingHelper list;
    InternetStackHelper internet;

    switch (m_protocol)
    {
    case 1:
        list.Add(olsr, 100);
        m_protocolName = "OLSR";
        break;
    case 2:
        list.Add(aodv, 100);
        m_protocolName = "AODV";
        break;
    case 3:
        list.Add(dsdv, 100);
        m_protocolName = "DSDV";
        break;
    case 4:
        m_protocolName = "DSR";
        break;
    default:
        NS_FATAL_ERROR("No such protocol:" << m_protocol);
    }

    if (m_protocol < 4)
    {
        internet.SetRoutingHelper(list);
        internet.Install(adhocNodes);
    }
    else if (m_protocol == 4)
    {
        internet.Install(adhocNodes);
        dsrMain.Install(dsr, adhocNodes);
    }
```

This code segment deals with different MANET Routing Protocols and Mobility models and sets their the values of the required parameters accordingly.

8 Reference

- **A Study of MANET Mobility Models**
  International Journal of Engineering and Technical Research (IJETR)
  ISSN: 2321-0869 (O) 2454-4698 (P), Volume-3, Issue-12,
  December 2015

9 Codes

- **Comparison of the MANET Routing Protocols and Mobility Models**
  Colab File
- **Routing Mobility**
  MANET Routing Protocols and Mobility Models
- **Random Walk**
  random walk code
- **Random Waypoint**
  random waypoint code