National Institute Of Technology, Hamirpur

Department Of Computer Science and Engineering

Mobile Computing Lab (CSD 427)
Practical Assignment 6

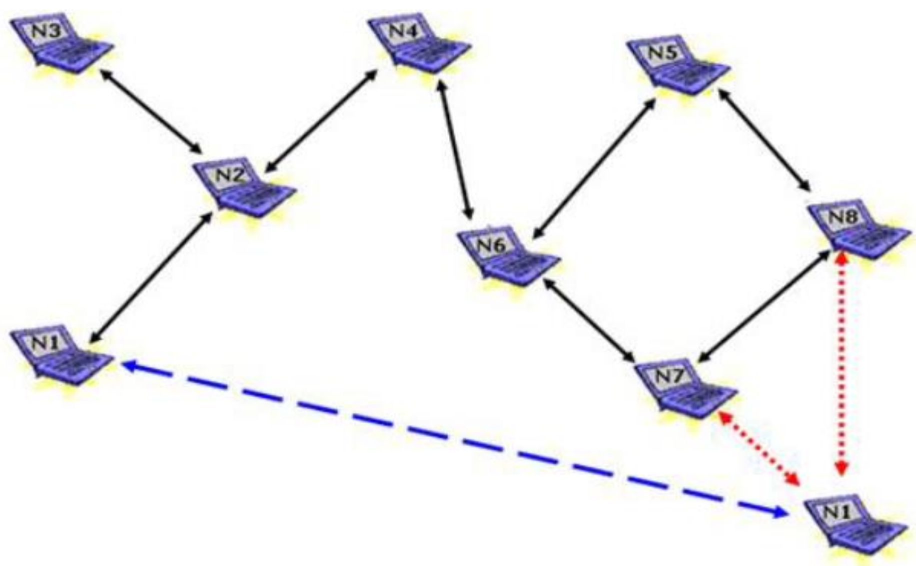
Submitted to:
Dr. Naveen Chauhan

Submitted by:
Jahnvi Gupta
Roll No. 17MI503
CSE Dual Degree

Aim: Write a Program for simulation of following routing protocols: DSR, DSDV, and AODV. Compare the performance of these protocols on various performance metrics.

DSDV Protocol:

DSDV (Destination-Sequenced Distance-Vector Routing Destination-Sequenced Distance-Vector Routing (DSDV) is a table-driven routing scheme for ad hoc mobile networks based on the Bellman–Ford algorithm. It was developed by C. Perkins and P.Bhagwat in 1994. The main contribution of the algorithm was to solve the routing loop problem. Each entry in the routing table contains a sequence number, the sequence numbers are generally even if a link is present; else, an odd number is used. The number is generated by the destination, and the emitter needs to send out the next update with this number. Routing information is distributed between nodes by sending full dumps infrequently and smaller incremental updates more frequently.



Code:

```
class Entry:
    def __init__(self, dest, next_hop, cost, seq_no):
        self.dest = dest
        self.next_hop = next_hop
        self.seq_no = seq_no
        self.cost = cost

    def __str__(self):
        return f'\t{self.dest},\t{self.next_hop},\t{self.cost},\t{self.seq_no}\t'
```

```python
class Node:
    def __init__(self, name):
        self.name = name
        self.seq_no = 0
        self.neighbors = []
        self.routing_table = {}


    def add_neighbor(self, node):
        self.neighbors.append(node)
        node.neighbors.append(self)

    def __str__(self):
        return f'{self.name}'


    def print_routing_table(self):
        print(f"----------------- {self.name}'s' routing table -------------------")
        print('    DestNode\tNextHop\tCost\tSeqNo')
        for node_entry in self.routing_table:
            print(f'{self.routing_table[node_entry]}')
        print(f'-------------------------------------------------------')

    def broadcast_table(self):
        self.routing_table[self.name].seq_no = self.seq_no
        self.seq_no += 2

        for node in self.neighbors:
            node.recieve_table(self.name, self.routing_table)

    def recieve_table(self, sender_name, sender_routing_table):

        made_update = False

        for key in self.routing_table.keys():
            sender_rountin_entry = sender_routing_table[key]
            own_routing_entry = self.routing_table[key]

            if(own_routing_entry.seq_no >= sender_rountin_entry.seq_no):
                continue

            # update the next entry
            if(own_routing_entry.cost > (sender_rountin_entry.cost + 1)):
                # make the update
                own_routing_entry.seq_no = sender_rountin_entry.seq_no
                own_routing_entry.dest = sender_rountin_entry.dest
                own_routing_entry.next_hop = sender_name
                own_routing_entry.cost = sender_rountin_entry.cost + 1
                made_update = True

        if made_update == True:
            self.broadcast_table()

INF = 1000


def print_routing_tables(network):
    for node in network:
        node.print_routing_table()
```

```
def main():
    A = Node("A")
    B = Node("B")
    C = Node("C")
    D = Node("D")

    A.add_neighbor(B)
    A.add_neighbor(C)
    C.add_neighbor(D)

    network = [A, B, C, D]

    print('----------   DSDV ALGORITHM   ----------')

    for node in network:
        # update the routing table of node to accomodate its own entry
        for other_node in network:

            if node == other_node:
                own_entry = Entry(node, node, 0, node.seq_no)

                node.routing_table[node.name] = own_entry

            else:
                init_entry = Entry(other_node, "", INF, -INF)

                node.routing_table[other_node.name] = init_entry


    print("BEFORE BROADCASTING")
    print_routing_tables(network)

    # A will start the broadcast
    A.broadcast_table()

    print("AFTER BROADCASTING")
    print_routing_tables(network)

if __name__ == '__main__':
    main()
```

Output:

```
----------   DSDV ALGORITHM   ----------
BEFORE BROADCASTING
----------------- A's' routing table -------------------
    DestNode    NextHop Cost    SeqNo
        A,      A,      0,      0
        B,      ,       1000,   -1000
        C,      ,       1000,   -1000
        D,      ,       1000,   -1000
--------------------------------------------------------
----------------- B's' routing table -------------------
    DestNode    NextHop Cost    SeqNo
        A,      ,       1000,   -1000
        B,      B,      0,      0
        C,      ,       1000,   -1000
        D,      ,       1000,   -1000
--------------------------------------------------------
----------------- C's' routing table -------------------
    DestNode    NextHop Cost    SeqNo
        A,      ,       1000,   -1000
        B,      ,       1000,   -1000
        C,      C,      0,      0
        D,      ,       1000,   -1000
--------------------------------------------------------
----------------- D's' routing table -------------------
    DestNode    NextHop Cost    SeqNo
        A,      ,       1000,   -1000
        B,      ,       1000,   -1000
        C,      ,       1000,   -1000
        D,      D,      0,      0
--------------------------------------------------------
```

Before Broadcasting

```
AFTER BROADCASTING
----------------- A's' routing table -------------------
    DestNode    NextHop Cost    SeqNo
        A,      A,      0,      6
        B,      B,      1,      0
        C,      C,      1,      0
        D,      C,      2,      0
--------------------------------------------------------
----------------- B's' routing table -------------------
    DestNode    NextHop Cost    SeqNo
        A,      A,      1,      0
        B,      B,      0,      4
        C,      A,      2,      0
        D,      A,      3,      0
--------------------------------------------------------
----------------- C's' routing table -------------------
    DestNode    NextHop Cost    SeqNo
        A,      A,      1,      2
        B,      A,      2,      0
        C,      C,      0,      2
        D,      D,      1,      0
--------------------------------------------------------
----------------- D's' routing table -------------------
    DestNode    NextHop Cost    SeqNo
        A,      C,      2,      2
        B,      C,      3,      0
        C,      C,      1,      0
        D,      D,      0,      0
--------------------------------------------------------
```

After Broadcasting

Advantages

The availability of paths to all destinations in network always shows that less delay is required in the path set up process The method of incremental update with sequence number labels, marks the existing wired network protocols adaptable to Ad-hoc wireless networks. Therefore, all available wired network protocol can be useful to ad hoc wireless networks with less modi cation

Disadvantages

DSDV requires a regular updates of its routing tables, which uses up battery power and a small amount of bandwidth even when the network is idle. Whenever the topology of the network changes, a new sequence number is necessary before the network re-converges; thus, DSDV is not suitable for highly dynamic or large scale networks. (As in all distance-vector protocols, this does not perturb traf c in regions of the network that are not concerned by the topology change.

2. DSR (Dynamic Source Routing)

The Dynamic Source Routing protocol (DSR) is a simple and efficient routing protocol designed speci cally for use in multi-hop wireless ad hoc networks of mobile nodes. DSR allows the network to be completely self-organizing and self-con guring, without the need for any existing network infrastructure or administration. It is a reactive protocol and all aspects of the protocol operate entirely on-demand basis. It works on the concept of source routing. Source routing is a routing technique in which the sender of a packet determines the complete sequence of nodes through which, the packets are forwarded. The advantage of source routing is : intermediate nodes do not need to maintain up to date routing information in order to route the packets they forward. The protocol is composed of the two main mechanisms of "Route Discovery" and "Route Maintenance". DSR requires each node to maintain a route – cache of all known self – to – destination pairs. If a node has a packet to send, it attempts to use this cache to deliver the packet. If the destination does not exist in the cache, then a route discovery phase is initiated to discover a route to destination, by sending a route request. This request includes the destination address, source address and a unique identification number. If a route is available from the route – cache, but is not valid anymore, a route maintenance procedure may be initiated. A node processes the route request packet only if it has not previously processes the packet and its address is not present in the route cache.

Code:

```
class Packet:
```

```
    def __init__(self, type, id, nodes, source, dest):
        self.type = type
        self.id = id
        self.path = nodes
        self.source = source
        self.dest = dest
    def get_path(self):
        path = [str(node) for node in self.path]

        return ' --> '.join(path)


class Node:
    def __init__(self, name):
        self.name = name
        self.neighbours = []
        self.recieved_packets = []
        self.path_cache = {}


    def add_neighbour(self, node):
        self.neighbours.append(node)
        node.neighbours.append(self)

    def is_packet_already_recieved(self, packet):
        return (packet.id in self.recieved_packets)

    def recieve_RREP(self, packet : Packet):
        print(f'Node {self.name} recieved RREP packet from {packet.source}')
        print(f'Path is : {packet.get_path()}')
        print('Adding path in path cache')
        self.path_cache[packet.dest] = packet.path
        print(f'Sending Data to {packet.source}')

    def send_RREQ(self, packet):
        print(f'Checking path cache')
        if packet.dest in self.path_cache.keys():
            print('Path already in cache... no need to send RREQ')
            return

        for node in self.neighbours:
            node.recieve_RREQ(packet)

    def recieve_RREQ(self, packet : Packet):
        print(f'Node {self} recieved RREQ packet')

        # check if the dest is not current node
        if packet.dest == self:
            print(f"Found Destination node... {self}")
            print(f'Node {self} is sending an RREP packet to {packet.source}')
            packet.path.append(self)
            packet.source.recieve_RREP(Packet(2, packet.id, packet.path, self, packet.source))
            return


        if self.is_packet_already_recieved(packet):
            return

        self.recieved_packets.append(packet.id)

        packet.path.append(self)
```

```python
        self.send_RREQ(packet)


    def __str__(self) -> str:
        return f'{self.name}'



def main():
    S = Node("S")
    A = Node("A")
    C = Node("C")
    D = Node("D")
    F = Node("F")

    S.add_neighbour(A)
    A.add_neighbour(C)
    C.add_neighbour(D)
    S.add_neighbour(D)
    D.add_neighbour(F)

    print('----------------------------------')
    print('---------  DRS PROTOCOL  ---------')

    print('----------------------------------')
    print('Sending Data from S to F')

    unique_id = 1
    S.recieved_packets.append(1)
    s_RREQ = Packet(1, unique_id, [S], S, F)
    S.send_RREQ(s_RREQ)


if __name__ == "__main__":
    main()
```

Output:

```
---------------------------------
---------  DRS PROTOCOL  ---------
---------------------------------
Sending Data from Node S to Node F
---------------------------------
Checking path cache
Node A recieved RREQ packet
Checking path cache
Node S recieved RREQ packet
Node C recieved RREQ packet
Checking path cache
Node A recieved RREQ packet
Node D recieved RREQ packet
Checking path cache
Node C recieved RREQ packet
Node S recieved RREQ packet
Node F recieved RREQ packet
Found Destination node... F
Node F is sending an RREP packet to S
Node S recieved RREP packet from F
Path is : S --> A --> C --> D --> F
Adding path in path cache
Sending Data to F
Node D recieved RREQ packet
```

Advantages

DSR uses a reactive approach which eliminates the need to periodically flood the network with table update messages which are required in a table-driven approach. The intermediate nodes also utilize the route cache information efficiently to reduce the control overhead

Disadvantages

The disadvantage of DSR is that the route maintenance mechanism does not locally repair a broken down link. The connection setup delay is higher than in table- driven protocols. Even though the protocol performs well in static and low-mobility environments, the performance degrades rapidly with increasing mobility. Also, considerable routing overhead is involved due to the source- routing mechanism employed in DSR. This routing overhead is directly proportional to the path length.

3. AODV Protocol:

Ad hoc On-Demand Distance Vector (AODV) Routing is a routing protocol for mobile ad hoc networks (MANETs) and other wireless ad hoc networks. It was jointly developed on July 2003 in Nokia Research Center, University of California, Santa Barbara and University of Cincinnati by C. Perkins, E. Belding-Royer and S. Das. AODV is the routing protocol used in ZigBee – a low power, low data rate wireless ad hoc network. There are various implementations of AODV such as MAD-HOC, Kernel-AODV, AODV-UU, AODV-UCSB and AODV- UIUC

Code:

```
class Packet:
    def __init__(self, source, source_seq, source_bcast_id, dest):
        self.source = source
        self.source_seq = source_seq
        self.source_bcast_id = source_bcast_id
        self.dest = dest
        self.dest_seq = ""
        self.hop_cnt = 0

    def copy(self):
        new_packet = Packet(self.source, self.source_seq, self.source_bcast_id, self.dest)
        new_packet.hop_cnt = self.hop_cnt + 1
        return new_packet



class Node:
    def __init__(self, name):
        self.name = name
        self.neighbors = []
        self.routing_table = {}
        self.seq_no = 0
        self.broadcast_id = 0
        self.recieved_packets = []

    def add_neighbor(self, node):
        self.neighbors.append(node)
        node.neighbors.append(self)

    def get_neighbours(self):
        neighbours = [str(node) for node in self.neighbors]
        return neighbours

    def broadcase_RREQ(self, packet):
        for node in self.neighbors:
            node.recieve_RREQ(self, packet)

    def print_routing_table(self):
        print(f'-------------------------------  {self} routing  --------------------------------')
        print(f'Dest\t    Next Hop\t    Hop Count\t    Seq. No')
        print(f'----------------------------------------------------------------------------')

        for key in self.routing_table.keys():
            entry = self.routing_table[key]
            nextRow = '\t\t'.join([str(node) for node in entry])
            print(f'{nextRow}')

        print(f'----------------------------------------------------------------------------')



    def entry_in_routing_table(self, dest):
        return str(dest) in self.routing_table.keys()
```

```python
def already_recieved_packet(self, packet):
    return packet.source_bcast_id in self.recieved_packets


def add_routing_entry(self, entry, sender):
    self.routing_table[entry.source.name] = [entry.source, sender, entry.hop_cnt, entry.source_seq]


def get_entry_from_routing_table(self, node):
    return self.routing_table[node.name]

def recieve_RREQ(self, sender, packet : Packet):
    print(f'Node {self} recieved RREQ from {sender}')

    if self.already_recieved_packet(packet):
        return


    self.recieved_packets.append(packet.source_bcast_id)

    new_packet = packet.copy()

    self.add_routing_entry(new_packet, sender)

    if self.entry_in_routing_table(packet.dest):
        print(f'Node {self} recieve RREQ from {sender}... Sending RREP to {packet.source}')
        return

    if new_packet.dest == self:
        print(f'{self} recieved the RREQ packet from {sender}.. Sending a reply to {sender}')

        RREP_Packet = Packet(self, self.seq_no, self.broadcast_id, packet.source)

        self.send_RREP(packet.source, RREP_Packet)
        return

    self.broadcase_RREQ(new_packet)

def send_RREP(self, node, packet):

    routing_table_entry = self.get_entry_from_routing_table(node)
    next_hop_node = routing_table_entry[1]

    next_hop_node.recieve_RREP(self, packet)

def recieve_RREP(self, sender, packet):
    print(f'Node {self} recieved RREP from {sender}.. meant for {packet.dest}')

    if self == packet.dest:
        print(f'Node {self} successfully recieved RREP from {packet.source}')
        return

    ## add an entry in table
    new_entry = packet.copy()

    self.add_routing_entry(new_entry, sender)
    self.send_RREP(new_entry.dest, new_entry)

def __str__(self) -> str:
```

```python
        return f'{self.name}'

def print_routing_tables(network):
    for node in network:
        node.print_routing_table()

        print("\n")


def main():
    A = Node("A")
    B = Node("B")
    C = Node("C")
    D = Node("D")
    E = Node("E")

    A.add_neighbor(B)
    B.add_neighbor(C)
    C.add_neighbor(D)
    E.add_neighbor(B)
    E.add_neighbor(C)
    network = [A, B, C, D, E]

    # A will start RREQ broadcase
    print_routing_tables(network)


    RREQ_Packet = Packet(A, A.seq_no, A.broadcast_id, E)
    print_routing_tables(network)


    A.add_routing_entry(RREQ_Packet, A)


    A.recieved_packets.append(RREQ_Packet.source_bcast_id)
    A.broadcase_RREQ(RREQ_Packet)

    print_routing_tables(network)



if __name__ == "__main__":
    main()
```

Output: **Before Routing**

```
------------------------------  A routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
----------------------------------------------------------------------------

------------------------------  B routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
----------------------------------------------------------------------------

------------------------------  C routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
----------------------------------------------------------------------------

------------------------------  D routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
----------------------------------------------------------------------------

------------------------------  E routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
----------------------------------------------------------------------------
```

Routing Procedure

```
Node B recieved RREQ from A
Node A recieved RREQ from B
Node C recieved RREQ from B
Node B recieved RREQ from C
Node D recieved RREQ from C
Node C recieved RREQ from D
Node E recieved RREQ from C
E recieved the RREQ packet from C.. Sending a reply to C
Node C recieved RREP from E.. meant for A
Node B recieved RREP from C.. meant for A
Node A recieved RREP from B.. meant for A
Node A successfully recieved RREP from E
Node E recieved RREQ from B
```

Final Routing Tables

```
------------------------------  A routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
A            A               0               0
----------------------------------------------------------------------------

------------------------------  B routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
A            A               1               0
E            C               2               0
----------------------------------------------------------------------------

------------------------------  C routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
A            B               2               0
E            E               1               0
----------------------------------------------------------------------------

------------------------------  D routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
A            C               3               0
----------------------------------------------------------------------------

------------------------------  E routing table ------------------------------
Dest        Next Hop        Hop Count       Seq. No
----------------------------------------------------------------------------
A            C               3               0
----------------------------------------------------------------------------
```

Advantages:

This protocol is reliable for the wireless mesh networks. AODV is loop free and does not require any cartelised system to handle routing process for wireless mesh networks

Disadvantages

Shortest path may be lost due to traffic during the path discovery process. AODV do not utilise any congestion control or avoidance mechanism to balance traffic load. The delivery ratio of AODV drops dramatically from more than 90% to about 28% when the number of connections increases from 10 to 50.