

Electronic, Electrical and Computer Engineering



UNIVERSITY OF
BIRMINGHAM

EE4L CCN

Computer and Communication Networks

Coursework Assignment 2015-2016

Dr A Feresidis

Andreas Oustas

1563901

MSc Electronic and Computer Engineering

Contents

1. Introduction.....	3
2. Part A	
2.1 Question 1.....	4
2.2 Question 2.....	6
3. Part B	
3.1 Description.....	10
3.2 Code Explanation.....	12
3.3 Results.....	16
3.4 Evaluation.....	19
4. Conclusion.....	20
5. Appendix	
5.1 Results.....	21
5.2 C# Code.....	24
6. References.....	28

1. Introduction

The main focus of this Computer and Communication Networks Assignment will be the routing protocols available for routing procedures and it will be divided into two different part.

In the first part there will be a discussion focused on routing protocols including distance vector and link state protocols and the main differences between the two of them will be highlighted and explained. Following that the Ad-hoc On-Demand Distance Vector (AODV) Routing Protocol will be expanded and explained thorough using diagrams and examples created from scratch for this purpose. The discussion will include route requests, reverse path setup, replies, data delivery as well as link failures and error handling.

In second part of this assignment a software implementation of a routing algorithm using flooding will be developed and explained using diagrams and the final results will be presented and discussed. Any possible solutions or obstacles found will also be mentioned.

Finally there will be a short conclusion summing up the assignment followed by the appendix which will include the code produced for this assignment as well as all the results gathered.

2. Part A

2.1 Question 1

What are the differences between a distance-vector and a link-state routing protocol? Describe at least two of them.

A router using the Distance vector protocol sends the full routing table, containing the distances to all available destinations, to all its neighbors meaning all the routers connected to it. On the other hand a router using the Link state protocol will only create a map of all the routers connected to it, meaning that each router constructs the network topology so in the end all the routers have the same information.

The main differences of the two protocols are:

- Each packet of data has unlimited hops when the Link state protocol is used but limited hops when the Distance vector protocol is used
- Usually based on the fact above, Distance vector protocol is used for home/rather small networks but Link state protocol is used for big networks with a large number of routers/nodes (Antun Peicevic. (2016))
- Distance vector protocol has high convergence time (slow) while Link state protocol has a low convergence time (fast)(Warren Heaton. (2000))
- When using the Distance vector protocol, routing loops can easily happen but when using the Link state protocol there are no routing loops
- When updating, the Distance vector protocol sends the entire routing table but the Link state protocol only sends updates when there is a change (Antun Peicevic. (2016))
- Distance vector protocol is easier to configure than Link state protocol
- Routers using the Distance vector protocol can't see the entire network while routers using the than Link state protocol have knowledge of the whole network
- Link state protocol requires more overhead (processing power and memory) than the Distance vector protocol (Warren Heaton. (2000))

Convergence Time:

Distance vector protocol uses a lot of bandwidth because it periodically sends the complete routing table to all the nodes in the network even if there is no change to the network causing high convergence time (CCNAStudyGuide. (2010)). Convergence time is the propagation time for the new information to reach all the nodes in a network. This high convergence time is the reason routing loops are common when using this protocol. Link state protocol on the other hand, as soon as it detects a change in the network, an event is triggered for the nodes to update their database (CCNAStudyGuide. (2010)). this results in very low convergence times thus reducing the chance of routing loops.

Overhead Difference – Ease of Use/Setup:

Link state protocol in order to calculate the best route for a router it uses all the information available from the neighbors and nearby nodes as well as databases stores in the routers. Distance vector protocol uses only information coming from the nodes connected to the router running the protocol (Andrew. (2011)). This extra work the Link state protocol does requires more processing power as well as more memory causing a network to be unstable if not properly configured that is why it is harder to implement. On the other hand Distance vector protocols use a lot of bandwidth (Andrew. (2011)), forcing the user to have at least an average network connection speed to properly implement this protocol, which is not necessary for the Link state protocol.

2.2 Question 2

What is the Ad-hoc On-Demand Distance Vector (AODV) Routing Protocol? In your description, use a diagram of your choice representing a network of at least ten wireless nodes (this should be different from the example of the lecture notes).

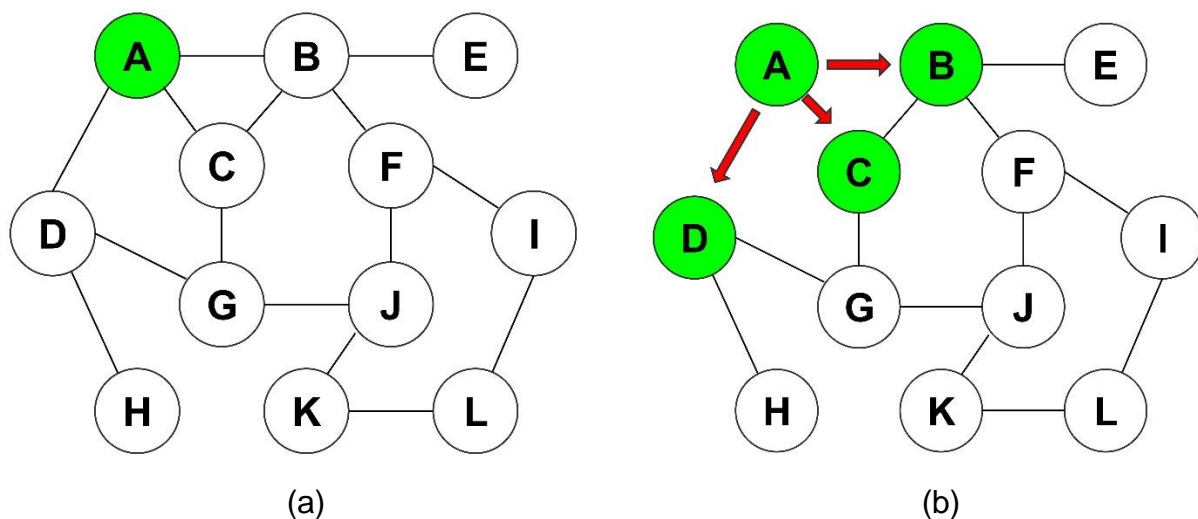
Discuss the following:

- Route request and reverse path setup
- Route reply and data delivery
- Link failure and route error handling

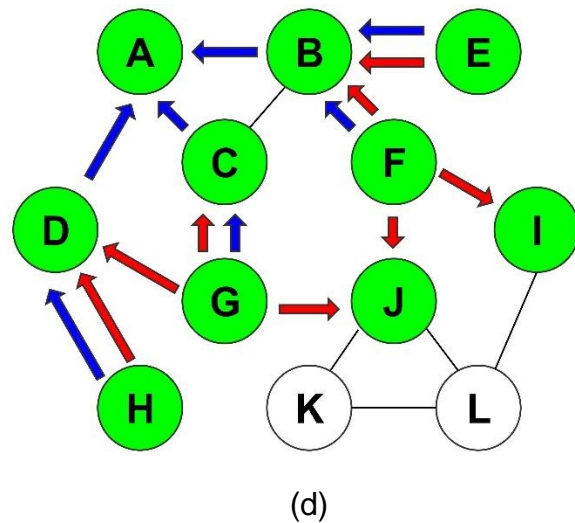
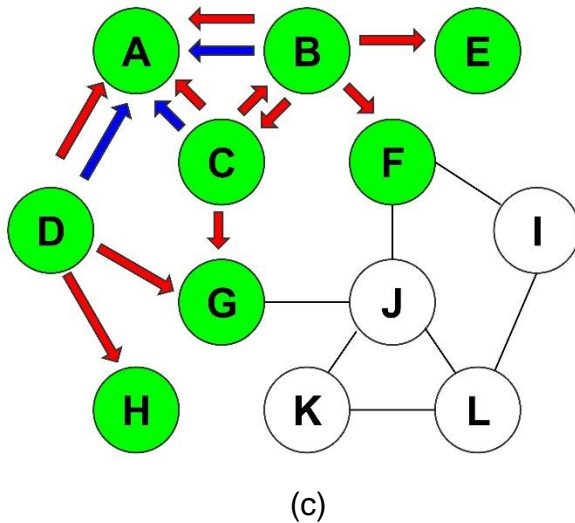
The Ad-hoc On-Demand Distance Vector routing protocol is a protocol used mainly for mobile or wireless mobile networks. The discovery process in the AODV protocol is similar to the process used by the Dynamic Source Routing protocol with the difference being that AODV dynamically sets and creates the routing tables in each node in between the start and the end of the path (Charles EP erkins) If one of the nodes stops responding in a path then only the nodes affected will be informed thus reducing the number of broadcasts need to be done in the network resulting in a smaller overhead.

There are 3 types of messages that the AODV protocol uses, Route Request (RREQs), Route Reply (RREPs) and Route Errors (RERRs).

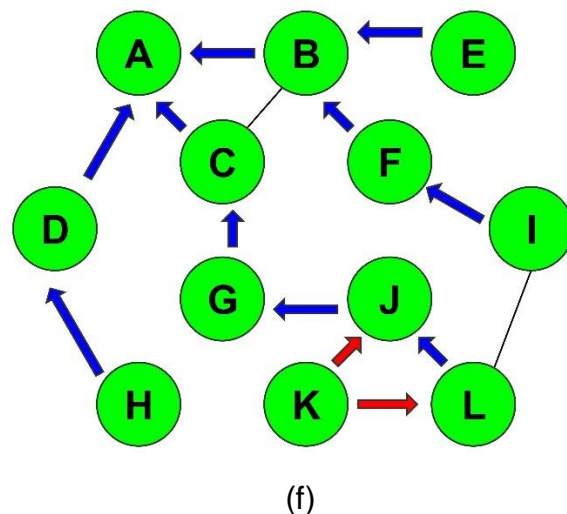
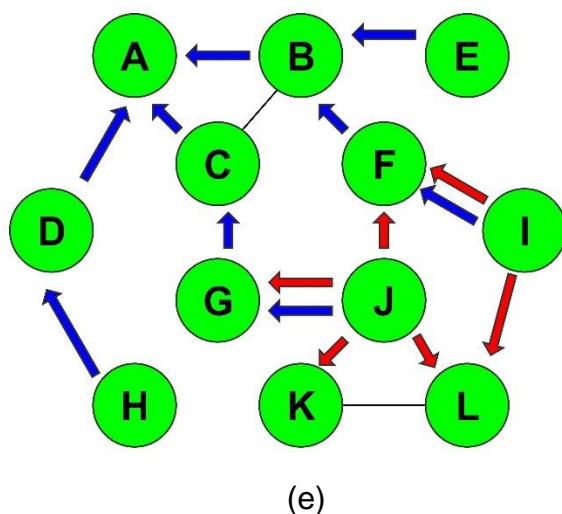
Route Request initializes the process for finding a route while each node that receives the RREQ creates a path back the origin node. When the request reaches the destination route then a RREP message is sent using the path discovered in order to confirm and finalize the path (Joe Barnes Chad Magers. (2003)). Route Error messages are sent in case there is a break between a node that is being used in an active path.



The first diagram (a) depicts a network made of 12 different nodes each connected to at least 1 other node in random order. The source node will be node A while the destination node used for this example will be node L meaning that the RREQ will be coming from node A to node L. In the second diagram (b) the discovery process has been initiated and the red arrows represent the transmission happening.

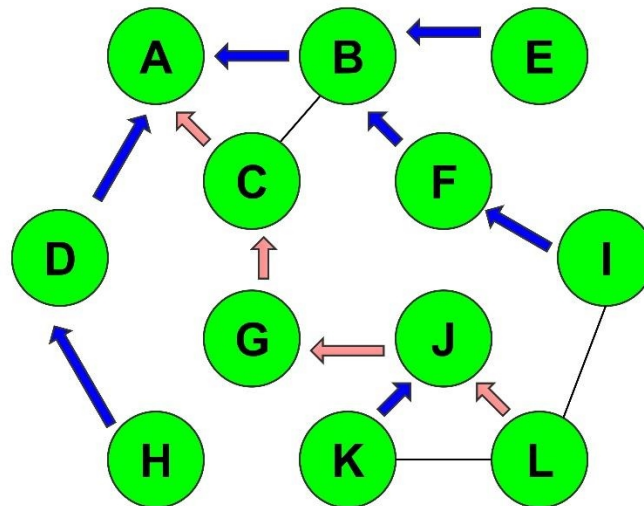


The next two diagrams represent the propagation of the RREQ up until one step before the destination node. The blue arrows depict the reverse path that will be used during the RREP and the actual data transfer. At any given time the network knows that if there is a request from node A to any other possible node the path is already known. Notice that at every step each node transmits to find out what is around it.



Diagrams 'e' and 'f' depict the final stage of the discovery having reached the destination node L. There is a chance, at some point during the request process, a node to receive a RREQ from two different neighbor nodes. In diagram 'e' this is what happens when node F receives two RREQ from node J and I, but due to the fact it has already transmitted a RREQ it won't do it again (Samir Das et al. (1999)).

Another thing worth mentioning is that if the RREQ has reached its destination, like in diagram 'f', the destination node L won't transmit the RREQ as it is the target node.



(g)

In the above diagram 'g' the pink arrows represent the path which the RREP will follow from the destination node L to the source node A. Note that in a network it is possible for a node, which is in-between the source and the destination node, to know a different path possibly more recent. In this case the destination sequence numbers method is used to deal with this possibility (Samir Das et al. (1999)). Each time a node receives a RREQ message, it increments its sequence number before transmitting the RREQ message so based on this number, if there is a shorter path recently created by adding a node to the network, the protocol will know that and it will choose the new path for the same transmission.

Timeouts are also implemented to avoid infinite RREQ and the duration of a timeout should be enough for a RREP message to reach the origin node. In case the timeout duration has been reached then the reverse path that has been established until that time will be deleted. Same method is followed also for the forward path but this time the timeout duration is set based on how long the user would like the path to be active without being used (Dr. Baruch Awerbuch & Dr. Amitabh Mishra).

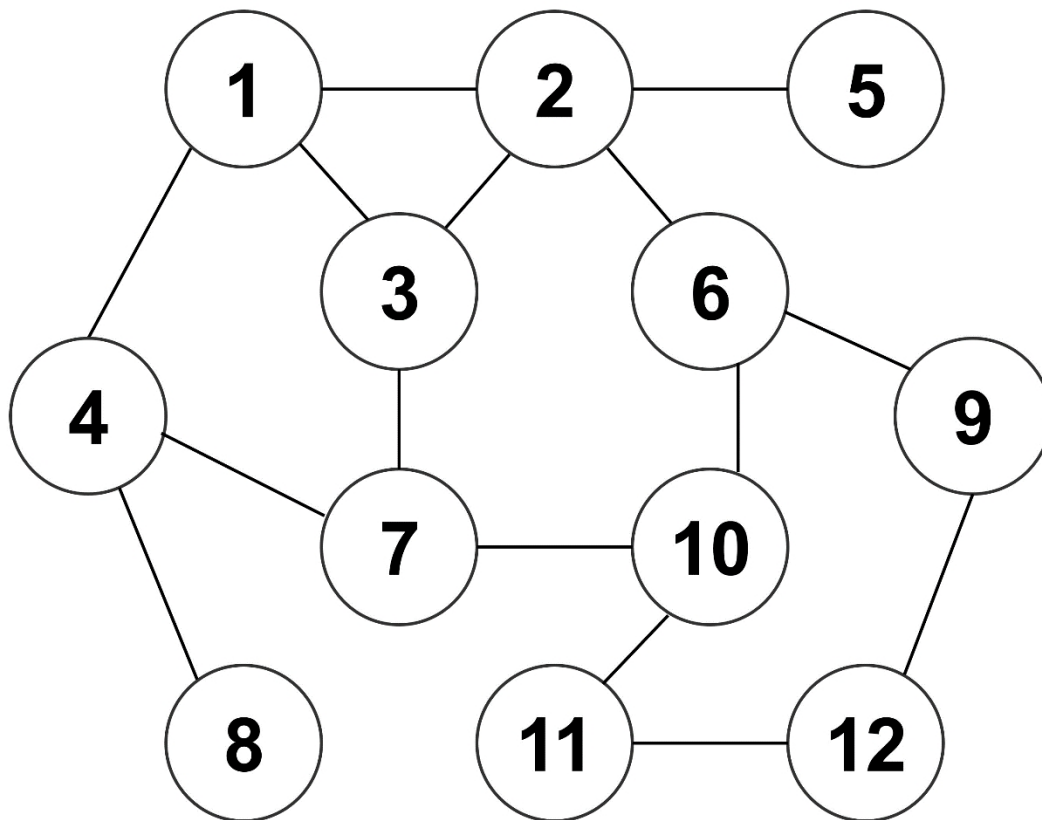


In case there is a link failure, as soon as it is detected, a Route Error message is sent by the nodes and at the same time the destination sequence numbers mentioned above (recent path) are updated. For the detection of a link failure another method is implemented. All the nodes, every now and then, send a short data packet, also called a hello message, to all the active neighbor nodes. If there is no response from a node or the response isn't complete, missing protocol address details, then it is automatically considered a link failure. Another case that can cause a RERR message is when a node receives a data packet that has a destination node that it is not active. For handling these route errors, first the node that detected the link failure marks the routing table entries that are connected with the error as invalid. Next all the nodes from this point increment their destination sequence numbers to the one in the RERR message and then continue the transmission. Finally when the RERR message reaches the origin node then this node will initiate a new route request message (RREQ). When initiating the new request, the destination sequence number already received with the RERR message is used to avoid following a route that will lead to the node that is inaccessible.

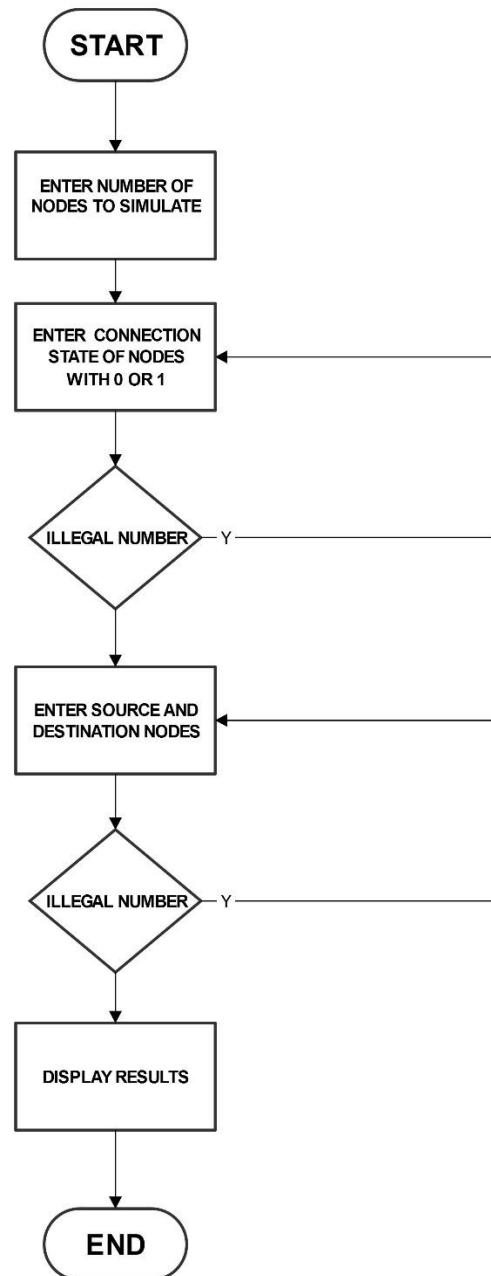
3. Part B

3.1 Description

For this part of assignment the C# programming language was used to simulate the flooding routing algorithm discussed in the lectures. The same network of nodes, used to complete the first of part of the assignment, was used but with a slight change in the naming of each node. In total there are 12 nodes interconnected randomly with each other.



As stated in the assignment brief, the code should have a hop counter for the packet being sent, each connection can only handle one packet per time and the testing should be executed using at least 10 node. Following the testing there should be a small discussion regarding the duplicate packets produced and any possible solution for minimizing this issue. Given the fact the same diagram from the first part is used, the testing will be executed using 12 nodes with node 1 as the source node and node 12 as the destination node.

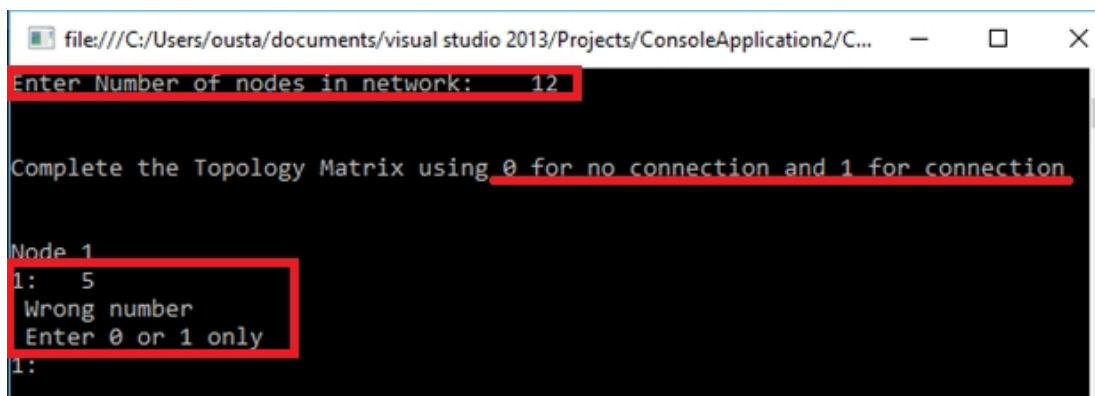


The above diagram depicts what it is expected from the program in terms of user interaction, meaning how the program's interface was constructed. At first the program will ask the user the total amount of nodes for the simulation. Following that the user will be asked to input which nodes are connected and which aren't by entering 1 if there is a connection and 0 if there is not. Finally the program will ask the user for the source and the destination node for the flooding to begin. The program will then display all the possible transmissions that can happen during a flooding.

3.2 Code Explanation

The program consists of a number of different functions, each one doing something different with the main section of the program being responsible for calling these functions and making sure everything runs smoothly by handling all the interrupts.

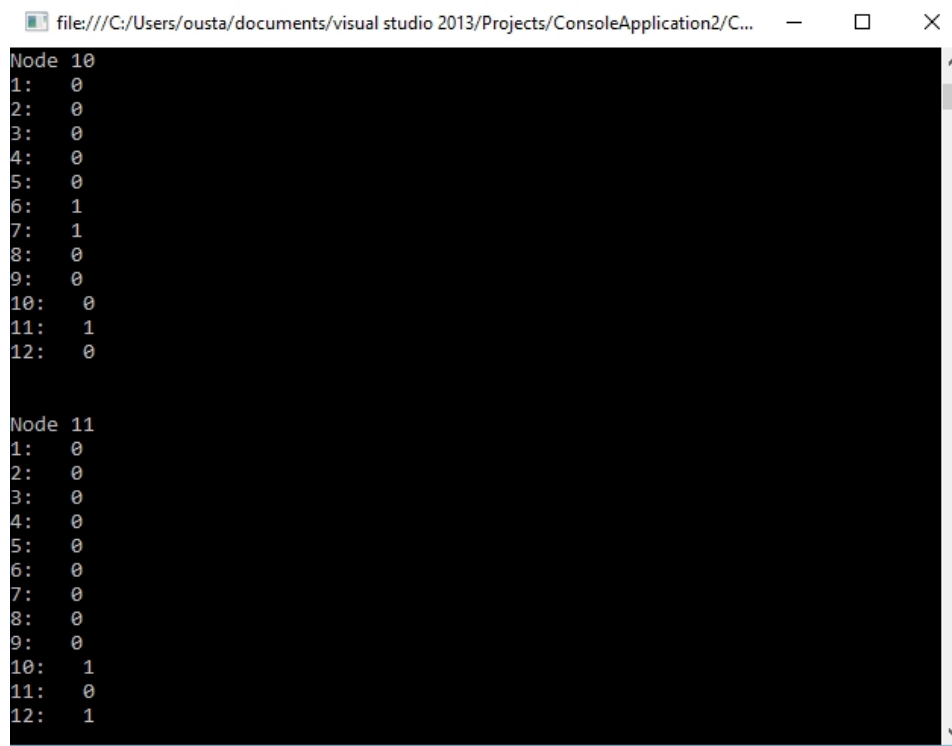
At first the main program asks the user of the number of nodes to be simulated and following that it asks for the user to enter the network topology by using 0 and 1 depending if there is a connection between the node or not. If, by mistake the user enters a different number than the one asked the program will cause an interrupt informing the user that he has entered an illegal number and to try again.



```
file:///C:/Users/ousta/documents/visual studio 2013/Projects/ConsoleApplication2/C...
Enter Number of nodes in network: 12

Complete the Topology Matrix using 0 for no connection and 1 for connection

Node 1
1: 5
Wrong number
Enter 0 or 1 only
1:
```



```
file:///C:/Users/ousta/documents/visual studio 2013/Projects/ConsoleApplication2/C...

Node 10
1: 0
2: 0
3: 0
4: 0
5: 0
6: 1
7: 1
8: 0
9: 0
10: 0
11: 1
12: 0

Node 11
1: 0
2: 0
3: 0
4: 0
5: 0
6: 0
7: 0
8: 0
9: 0
10: 1
11: 0
12: 1
```

Node 10 and 11 already setup with their connections in regards to the other nodes.

After the main network matrix has been filled with the topology of the network, the first function is called. The Topology function will simply print out the matrix containing ones and zeroes for the user to check if he entered them correctly.

```
file:///C:/Users/ousta/documents/visual studio 2013/Projects/ConsoleApplication2/C... - □ X
```

Network Matrix 1st row and colomn showing nodes(or hops) id

Nodes	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	1	0	0	0	0	0	0	0	0
2	1	0	1	0	1	1	0	0	0	0	0	0
3	1	1	0	0	0	0	1	0	0	0	0	0
4	1	0	0	0	0	0	1	1	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	1	1	0	0
7	0	0	1	1	0	0	0	0	0	1	0	0
8	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0	0	0	1
10	0	0	0	0	0	1	1	0	0	0	1	0
11	0	0	0	0	0	0	0	0	0	1	0	1
12	0	0	0	0	0	0	0	0	1	0	1	0

Maximum Hops needed to reach any destination from any source : 9

Next a function called Hops is called by the main program to calculate the maximum number of hops needed for a packet to reach any destination from any source. Note that this is done automatically to avoid any packet loss but if needed for the purposes of the simulation, it can be changed to whatever number the user wants and see what the result will be. The algorithm for calculating the hops is simple but effective. First it sums up all the connections based on the topology and subtracts the number of nodes in the matrix. If this number is an even number then the result gets divided by 2 and if it is odd then it gets divided by 3.

```
life -= n;

if ((life % 2) == 0) // condition for odd or even to decide hop count
    life = life / 2;
else
    life= life / 3;
return life;
```

Following that the user is asked to enter the source and the destination nodes for the flooding to take place. Same as before, there is a check whether the user entered a

legal number, meaning that if the user enters a number that does not correspond to a node then the program will inform him and ask him to enter the number again.

```
file:///C:/Users/ousta/documents/visual studio 2013/Projects/ConsoleApplication2/C...
Network Matrix 1st row and colomn showing nodes(or hops) id
Nodes    1    2    3    4    5    6    7    8    9   10   11   12
1        0    1    1    1    0    0    0    0    0    0    0    0
2        1    0    1    0    1    1    0    0    0    0    0    0
3        1    1    0    0    0    0    1    0    0    0    0    0
4        1    0    0    0    0    0    1    1    0    0    0    0
5        0    1    0    0    0    0    0    0    0    0    0    0
6        0    1    0    0    0    0    0    0    1    1    0    0
7        0    0    1    1    0    0    0    0    0    1    0    0
8        0    0    0    1    0    0    0    0    0    0    0    0
9        0    0    0    0    0    1    0    0    0    0    0    1
10       0    0    0    0    0    1    1    0    0    0    1    0
11       0    0    0    0    0    0    0    0    0    1    0    1
12       0    0    0    0    0    0    0    0    1    0    1    0

Maximum Hops needed to reach any destination from any source : 9

Enter the Source Node number: 1
Enter the Destination Node number: 12
```

The last part of the program is the actual flooding algorithm function which is called after all the above information have been entered. The flooding function first prints out the origin node and proceeds to call the floodcheck function which will flood the nodes connected the origin node and keep flooding the neighbor nodes until it reaches the destination node or the hop counter reaches 0. As this function floods the network it continuously prints out the flood nodes and their neighbors.

```
file:///C:/Users/ousta/documents/visual studio 2013/Projects/ConsoleApplication2/C...
Enter the Source Node number: 1
Enter the Destination Node number: 12
1  ORIGIN NODE
First Results
2  3  4
3  5  6
1  7
2  4
3  5  6
```

As soon as the destination node is entered then the first results start to appear. The results go on until one of the two conditions is met.

3.3 Results

After the program executes, it displays the results to the user and waits until any key is pressed for the console window to close. Below are the results of the flooding algorithm starting from node 1 to node 12. For the sake of the test, the condition for reaching the destination node has been altered, allowing for all the possible node combinations, due to the flooding, to be observed.

```

Enter the Source Node number: 1

Enter the Destination Node number: 12

1

2 3 4

3 5 6

1 7

```

The picture above depicts the first flooding that has already happened from node 1. Based on the network topology already discussed, node 1 floods and its neighbor nodes receive the packet, node 2, 3 and 4. Then node 2 transmits the packet and now its neighbors receive it, nodes 3, 5 and 6. This process goes on until all nodes are covered but it doesn't stop upon first reaching the destination node 12 as this condition has been removed so the whole flooding can be observed. All the results can be found in the appendix with comments. Note that, for ease of use, the results have been transferred into an excel file.

	A	B	C	D	E
1	Nodes the packet has reached			Comments	
2	1			Origin Node	
3	2	3	4	Node 1 Neighbors	
4	3	5	6	Node 2 Neighbors	
5	1	7	8	Node 4 Neighbors	
6	2	4		Cycling back to node 1	
7	3	5	6	Duplicate packets	
8	1	7			
9	2	4			
10	3	5	6		
11	1	7			
12					
13					
14	9	10		Node 6 Neighbors	
15	7	8		Node 4 Neighbors	
16	3	10		Node 7 Neighbors	
17					
18					
19	4	10		Node 7 Neighbors	
20	1	8		Node 4 Neighbors	
21	2	3		Node 1 Neighbors	
22					
23					
24	6	11		Node 10 Neighbors	
25	2	9		Node 6 Neighbors	
26	12			Destination Node	

28				
29	9	10		Node 6 Neighbors
30	12			
31	7	11		Node 10 Neighbors
32	3	4		Node 1 Neighbors
33	1	2		Node 1 Neighbors
34	1	8		Node 4 Neighbors
35	12			Destination Node
36	7	8		Node 4 Neighbors
37	3	10		Node 7 Neighbors
38	1	2		Node 3 Neighbors
39	2	4		Node 1 Neighbors
40	3	5	6	Node 2 Neighbors
41	7	8		Node 4 Neighbors
42	1	5	6	Node 2 Neighbors
43	3	4		Node 1 Neighbors
44				
45				
46	9	10		Node 6 Neighbors
47	6	11		Node 10 Neighbors
48	2	9		Node 6 Neighbors
49	1	3	5	Node 2 Neighbors
50	12			Destination Node
51	12			
52				
53				
54	4	10		Node 7 Neighbors
55	1	8		Node 4 Neighbors
56	2	3		Node 1 Neighbors
57	3	5	6	Node 2 Neighbors
58	1	7		Node 4 Neighbors
59	2	4		Node 1 Neighbors
60	4	10		Node 7 Neighbors

63	9	10		Node 6 Neighbors
64	12			Destination Node
65	7	11		Node 10 Neighbors
66	2	7		Node 3 Neighbors
67	1	5	6	Node 2 Neighbors
68	3	4		Node 1 Neighbors
69				
70				
71	9	10		Node 6 Neighbors
72	4	10		Node 7 Neighbors
73	1	8		Node 4 Neighbors
74	6	11		Node 10 Neighbors
75				
76				
77	6	11		Node 10 Neighbors
78	2	9		Node 6 Neighbors
79	1	3	5	Node 2 Neighbors
80	3	4		Node 1 Neighbors
81	2	7		Node 3 Neighbors
82	7	8		Node 4 Neighbors
83	1	7		Node 4 Neighbors
84	2	4		Node 1 Neighbors
85	4	10		Node 7 Neighbors
86				
87				
88	12			Destination Node
89	12			
90				
91				
92	9	10		Node 6 Neighbors
93	12			Destination Node
94	7	11		Node 10 Neighbors
95	3	4		Node 1 Neighbors
96	1	2		Node 3 Neighbors
97	2	4		Node 1 Neighbors
98	3	5	6	Node 2 Neighbors
99	1	7		Node 4 Neighbors

The first few results until the destination node has been reached can be observed above. Starting with node 1 as the origin node the packet spreads based on the topology of the initial diagram. Node 12 is reached multiple times with the first being during the 4th hop of the algorithm. The spread continues for each individual node and the packet reaches each the node at least 6 times up until the point shown above. As the results continue to appear on the screen while the algorithm runs, every node appears on the list and eventually the results show all the possible transmission that can happen but with at least 15 duplicates packets been received from each node in the network. This is clearly an issue and possible solution will be suggested and discussed later on, in the evaluation part of this report.

Another example is the following where the destination node has been reached again.

93	9	10		Node 6 Neighbors
94	12			Destination Node
95	7	11		Node 10 Neighbors
96	3	4		Node 1 Neighbors
97	1	2		Node 3 Neighbors
98	2	4		Node 1 Neighbors
99	3	5	6	Node 2 Neighbors
100	1	7		Node 4 Neighbors

Eventually the packet reaches node 6. From there it spreads to node 9 and 10 and from 9 it reaches node 12. From node 10 it spreads to node 7 and 11 and so on until it loops back to node 4 and node 1 again.

55	4	10		Node 7 Neighbors
56	1	8		Node 4 Neighbors
57	2	3		Node 1 Neighbors
58	3	5	6	Node 2 Neighbors
59	1	7		Node 4 Neighbors
60	2	4		Node 1 Neighbors
61	4	10		Node 7 Neighbors

Same loop can be observed in the above screenshot but starting with node 7, the program loops back to it through node 4, 1 and 2.

Based on the results it can be said that the program does work as intended and each line/connection between the nodes can only send data in one direction at a time. Also the above result shows that if the hop is limited to a specific number the packet will only reach a specific node and then the program will quit. The issue identified is the issue with the duplicate packets and it will be discussed later on.

3.4 Evaluation

As seen in the results section there are a lot of duplicate packets reaching the nodes constantly and this is due to the initial construction of the program and the flooding algorithm. This should be avoided in real life scenarios as it will causes problems to the network in terms of speed, security and stability of the overall topology. A possible solution to this problem in the code, will be to modify the flooding algorithm to not need to use the topology matrix two times thus enabling the programmer to erase the matrix by placing 0 in the place of 1 (active connections) so the next time to function tries to go over the matrix it won't find any connection, meaning that the node has already received the packet resulting in the algorithm to bypass it.

Another possible solution can be another matrix, independent of the flooding, which will get updated every time a node receives a packet. This can be done by enabling the nodes to individually send a message containing a confirmation that they already received a packet along with their unique ID. The matrix will then contain only the nodes that have yet to be flooded so that it can be used by the flooding function to continue its flooding operation.

4. Conclusion

In this assignment a small summary and differences between the distance-vector and the link-state routing protocols have been given and discussed along with a more elaborate analysis on a few of them. Next there was an extensive discussion around the Ad-hoc On-Demand Distance Vector routing protocol mainly focusing on Route request and reverse path setup, Route reply and data delivery, Link failure and route error handling. This discussion was supported by an example created from scratch for this purpose and all the information used were gathered by a number of sources as stated and included in the appendix section. All diagrams have been created using an online, free to use, drawing tool.

Following the theory questions came the second part of the assignment. In this part the creation of a program for simulating routing using flooding in a network of 12 nodes was implemented successfully. All the parts of the code have been explained individually and the parts were supported by screenshots taken directly from the compiler used to create the program. Next the results were explained using screenshots from the console running the code and any problems encountered were mentioned. These problems were then explained in more detail and 2 possible solution were suggested for solving them.

In the end, the completion of this assignment was successful and the objectives stated in the assignment brief have been reached.

5. Appendix

Results

Nodes the packet has reached			Comments	Nodes the packet has reached			Comments
1			Origin Node	9	10		Node 6 Neighbors
2	3	4	Node 1 Neighbors	12			Destination Node
							Node 10
3	5	6	Node 2 Neighbors	7	11		Neighbors
1	7	8	Node 4 Neighbors	1	8		Node 4 Neighbors
2	4		Cycling back to node 1	2	3		Node 1 Neighbors
3	5	6	Duplicate packets	3	5	6	Node 2 Neighbors
1	7			1	7		Node 4 Neighbors
2	4						
3	5	6					
1	7			9	10		Node 6 Neighbors
				2	7		Node 3 Neighbors
				1	5	6	Node 2 Neighbors
9	10		Node 6 Neighbors	4	10		Node 7 Neighbors
7	8		Node 4 Neighbors				
3	10		Node 7 Neighbors				
				12			Destination Node
				2	7		Node 3 Neighbors
4	10		Node 7 Neighbors	1	5	6	Node 2 Neighbors
1	8		Node 4 Neighbors	3	4		Node 1 Neighbors
2	3		Node 1 Neighbors	2	7		Node 2 Neighbors
				1	5	6	Node 2 Neighbors
				3	4		Node 1 Neighbors
6	11		Node 10 Neighbors	2	7		Node 3 Neighbors
2	9		Node 6 Neighbors	1	5	6	Node 2 Neighbors
12			Destination Node	4	10		Node 7 Neighbors
				7	8		Node 4 Neighbors
				3	10		Node 7 Neighbors
9	10		Node 6 Neighbors				
12				9	10		Node 6 Neighbors
7	11		Node 10 Neighbors				
3	4		Node 1 Neighbors	12			Destination Node
							Node 10
1	2		Node 1 Neighbors	7	11		Neighbors
1	8		Node 4 Neighbors	3	4		Node 1 Neighbors

12			Destination Node
7	8		Node 4 Neighbors
3	10		Node 7 Neighbors
1	2		Node 3 Neighbors
2	4		Node 1 Neighbors
3	5	6	Node 2 Neighbors
7	8		Node 4 Neighbors
1	5	6	Node 2 Neighbors
3	4		Node 1 Neighbors
9	10		Node 6 Neighbors
6	11		Node 10 Neighbors
2	9		Node 6 Neighbors
1	3	5	Node 2 Neighbors
12			Destination Node
12			
4	10		Node 7 Neighbors
1	8		Node 4 Neighbors
2	3		Node 1 Neighbors
3	5	6	Node 2 Neighbors
1	7		Node 4 Neighbors
2	4		Node 1 Neighbors
4	10		Node 7 Neighbors
9	10		Node 6 Neighbors
12			Destination Node
7	11		Node 10 Neighbors
2	7		Node 3 Neighbors
1	5	6	Node 2 Neighbors
3	4		Node 1 Neighbors
9	10		Node 6 Neighbors
4	10		Node 7 Neighbors
1	8		Node 4 Neighbors
6	11		Node 10 Neighbors

12			Destination Node
4	10		Node 7 Neighbors
1	8		Node 4 Neighbors
2	3		Node 1 Neighbors
3	5	6	Node 2 Neighbors
2	7		Node 3 Neighbors
6	11		Node 10 Neighbors
2	9		Node 6 Neighbors
1	3	5	Node 2 Neighbors
12			Node 7 Neighbors
12			
7	8		Node 4 Neighbors
3	10		Node 7 Neighbors
1	2		Node 3 Neighbors
2	4		Node 1 Neighbors
3	5	6	Node 2 Neighbors
7	8		Node 4 Neighbors
1	5	6	Node 2 Neighbors
3	4		Node 1 Neighbors
9	10		Node 6 Neighbors
6	11		Node 10 Neighbors
2	9		Node 6 Neighbors
1	3	5	Node 2 Neighbors
12			Destination Node
12			
9	10		Node 6 Neighbors
12			Destination Node
7	11		Node 10 Neighbors
2	7		Node 3 Neighbors
1	5	6	Node 2 Neighbors
3	4		Node 1 Neighbors
7	11		Node 10 Neighbors
3	4		Node 1 Neighbors
1	2		Node 3 Neighbors
2	4		Node 1 Neighbors
3	5	6	Node 2 Neighbors
7	8		Node 4 Neighbors
1	5	6	Node 2 Neighbors
3	4		Node 1 Neighbors

6	11		Node 10 Neighbors			
2	9		Node 6 Neighbors	9	10	Node 6 Neighbors
1	3	5	Node 2 Neighbors	1	8	Node 4 Neighbors
3	4		Node 1 Neighbors	2	3	Node 1 Neighbors
2	7		Node 3 Neighbors	3	5	6 Node 2 Neighbors
7	8		Node 4 Neighbors	2	7	Node 3 Neighbors
1	7		Node 4 Neighbors			
2	4		Node 1 Neighbors			
4	10		Node 7 Neighbors	12		Destination Node
				4	10	Node 7 Neighbors
				1	8	Node 4 Neighbors
12			Destination Node	2	3	Node 1 Neighbors
12				3	5	6 Node 2 Neighbors
				1	7	Node 4 Neighbors
				2	4	Node 1 Neighbors
9	10		Node 6 Neighbors	3	5	6 Node 2 Neighbors
12			Destination Node	7	8	Node 4 Neighbors
7	11		Node 10 Neighbors	4	10	Node 7 Neighbors
3	4		Node 1 Neighbors	1	8	Node 4 Neighbors
1	2		Node 3 Neighbors	6	11	Node 10 Neighbors
2	4		Node 1 Neighbors			
3	5	6	Node 2 Neighbors			
1	7		Node 4 Neighbors	9	10	Node 6 Neighbors
				12		Destination Node
				7	11	Node 10 Neighbors
9	10		Node 6 Neighbors	3	4	Node 1 Neighbors
7	8		Node 4 Neighbors	12		Destination Node
3	10		Node 7 Neighbors	2	7	Node 3 Neighbors
				1	5	6 Node 2 Neighbors
				3	4	Node 1 Neighbors
1	5	6	Node 2 Neighbors	2	7	Node 3 Neighbors
3	4		Node 1 Neighbors	7	8	Node 4 Neighbors
2	7		Node 3 Neighbors			
7	8		Node 4 Neighbors			

C# code

```
using System;

public class FloodingAlgo
{
    //Hops function to decide the TTL of the packet in the network network
    internal static int Hops(int[][] m, int n)
    {
        int life = 0;
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                life += m[i][j];
            }
        }
        life -= n;

        if ((life % 2) == 0) // condition for odd or even to decide hop count
            life = life / 2;
        else
            life = life / 3;
        return life;
    }

    //Topology function displays the network topology in a matrix form as
    entered by the user
    public static void Topology(int[][] m, int n)
    {
        Console.WriteLine("\n\n Network Matrix \n\n");
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= n; j++)
            {
                if (i == 0 && j == 0)
                {
                    Console.Write("Nodes      "); //Title
                }
                else if (j == 0)
                {
                    Console.Write(m[i][j] + "      ");
                }
                else
                {
                    Console.Write(m[i][j] + "    ");
                }
            }
            Console.WriteLine();
        }
    }

    internal static void Flood(int[][] m, int n, int source, int destination,
int life)
    {

```



```

        Console.WriteLine("\n" + source); //prints out the origin node
        Console.WriteLine("\n");
        floodcheck(m, n, 0, source, destination, life); //floodcheck function
        floods the next node                                //it keeps repeating until the
                                                            //is reached or Hops counter becomes
destination node
zero;
    }

    internal static void floodcheck(int[][] m, int n, int previous, int next,
int destination, int life)
    {
        //m = Topology Matrix , n = Total Nodes , previous = Previous Node
        , next = Next Node , destination = Destination Node , life = Life of Data -
        Remaining Hops
        int i = next;
        if (life == 0 || next == destination) //condition to stop the
function
        {
            return;
        }
        for (int j = 1; j <= n; j++)
        {
            if (m[i][j] == 1 && j != i && j != previous)
            {
                Console.Write(j + " "); // prints out the flooded nodes
            }
        }
        Console.WriteLine("\n\n"); //double enter space
        Console.WriteLine("\n\n");
        for (int j = 1; j <= n; j++)
        {
            if (m[i][j] == 1 && j != i && j != previous) //same condition
back used to recall floodcheck again
            {
                floodcheck(m, n, next, j, destination, (life - 1)); //
floodcheck function called inside it

//manipulating this will cause different results
                                                                    // change
life to reduce hops - dropped packets
            }
        }
    }

    public static void Main(string[] args)
    {
        Console.Write("Enter Number of nodes in network: ");

        int n ;
        string flag3 = Console.ReadLine(); //reads number of nodes
        Int32.TryParse(flag3, out n);
    }
}

```

```

        //1 stands for connection and 0 stands for no connection between two
nodes
        Console.WriteLine("\n");
        Console.WriteLine("Complete the Topology Matrix using 0 for no
connection and 1 for connection");

        int[][] network = RectangularArrays.ReturnRectangularIntArray(n + 1,
n + 1); //declaration for using a 2d matrix

        //first column and row used to save the node ids
        for (int i = 1; i <= n; i++)
        {
            network[0][i] = i;
            network[i][0] = i;
        }

        //2d matrix filled with the network topology
        for (int i = 1; i <= n; i++)
        {
            Console.WriteLine("\n");
            Console.WriteLine("Node " + (i));
            Console.WriteLine("\n");
            for (int j = 1; j <= n; j++)
            {
                Console.Write((j) + ": ");
                string flag2 = Console.ReadLine(); // read 1 or 0
                int c;
                Int32.TryParse(flag2, out c);

                if (c == 0 || c == 1)
                {
                    network[i][j] = c;
                }
                else
                {
                    Console.WriteLine(" Wrong number\n Enter 0 or 1
only"); //check for illegal number
                    j -= 1;
                }
            }
        }

        Topology(network, n); // function called to print out the 2d matrix

        int life = Hops(network, n); // life variable for the maximum ttl of
the packet
        Console.WriteLine("\n");
        Console.WriteLine("Maximum Hops needed to reach any destination from
any source : " + (life));
        Console.WriteLine("\n");

        int flag; // variable used to determine error
        string input;
        do
        {

```

```

        Console.WriteLine("\n");
        Console.Write(" Enter the Source Node number:  ");

        input = Console.ReadLine(); //reads the source node
        Int32.TryParse(input, out flag);

        if (flag == 0 || flag > n)
        {
            Console.Write("No such node exists\n Try again");// exception
handler
        }

    } while (flag == 0 || flag > n);

    int source = flag; // source variable saves the source node id
    do
    {
        Console.WriteLine("\n");
        Console.Write(" Enter the Destination Node number:  ");

        input = Console.ReadLine();
        Int32.TryParse(input, out flag);

        if (flag == 0 || flag > n)
        {
            Console.Write("Entered wrong id number not available in
network.try again.\n Re");// exception handler
        }

    }while (flag == 0 || flag > n);

    int destination = flag; // destination variable saves the destination
node id
    Flood(network,n,source,destination,life); // flooding function called
for the flooding to start
    Console.ReadLine();
}

}

internal static class RectangularArrays // helper function for the creation
of a 2d matrix
{
    internal static int[][] ReturnRectangularIntArray(int size1, int size2)
    {
        int[][] newArray = new int[size1][];
        for (int array1 = 0; array1 < size1; array1++)
        {
            newArray[array1] = new int[size2];
        }

        return newArray;
    }
}

```

6. References

Warren Heaton. (2000). *Should you use distance vector or link state routing protocols?*. Available: <http://www.techrepublic.com/article/should-you-use-distance-vector-or-link-state-routing-protocols/>. Last accessed 02/05/2016.

Antun Peicevic. (2016). *Routing protocols* . Available: <http://study-ccna.com/routing-protocols/>. Last accessed 02/05/2016.

Andrew. (2011). *Difference Between Link State and Distance Vector*. Available: <http://www.differencebetween.com/difference-between-link-state-and-vs-distance-vector/>. Last accessed 02/05/2016.

CCNAStudyGuide. (2010). *Distance vector and link state routing protocol*. Available: <http://www.slideshare.net/CCNAStudyGuide/distance-vector-and-link-state-routing-protocol>. Last accessed 02/05/2016.

Charles E Perkins, Elizabeth M Royer. (-). *Ad hoc On Demand Distance Vector Routing*. Available: <https://www.cs.cornell.edu/people/egs/615/aodv.pdf>. Last accessed 02/05/2016.

Joe Barnes Chad Magers. (2003). *Ad hoc On - Demand Distance Vector (AODV) Routing*. Available: <http://www.cs.uah.edu/~fzhu/570/paper4.pdf>. Last accessed 02/05/2016.

Samir Das et.. (1999). *Ad-Hoc On-Demand Distance Vector (AODV) Routing*. Available: https://www.researchgate.net/publication/221611438_Ad-hoc_On-Demand_Distance_Vector_Routing. Last accessed 02/05/2016.

Dr. Baruch Awerbuch & Dr. Amitabh Mishra. (-). *4-1 Ad hoc On Demand Distance Vector (AODV) Routing Protocol*. Available: <http://www.cs.jhu.edu/~cs647/aodv.pdf>. Last accessed 02/05/2016.