

Tia Blansett, Abir Mojumder, and Loken Vande Vegte

Dr. Hongwei Zhang

CPR E 537

12 May 2022

Contiki-Cooja Simulator: RPL-UDP Rank (Sinkhole) Attack and Detection via IDS

Attack

The RPL Sinkhole attack uses the ability of DIO (DODAG Information Object) Rank advertisement to fool neighboring nodes to send packets to the attacker instead of a benign node. The code modification is done within the RPL-ipv6 file within the Contiki OS files. The change is a false advertisement of rank as shown in Figure 1.

```
void
dio_output(rpl_instance_t *instance, uip_ipaddr_t *uc_addr)
{
    unsigned char *buffer;
    int pos;
    rpl_dag_t *dag = instance->current_dag;
    dag->rank = ROOT_RANK(instance);
    PRINTF("Advertising rank %u\n", dag->rank);
}
```

Figure 1. Node rank advertisement code.

In the code, the rank is changed to `ROOT_RANK(instance)`. This changes the node's own rank, to the local (within TX range) root node's rank. The packets are supposed to go from a node of higher rank to a node of lower rank as shown in Figure 2.

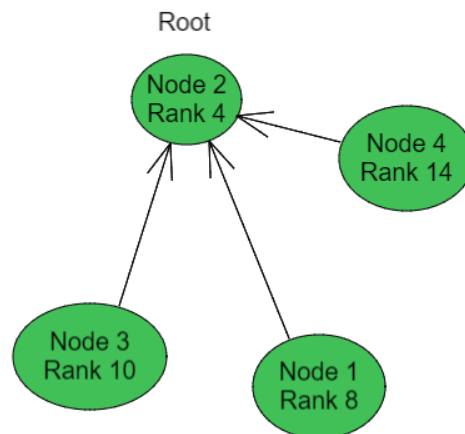


Figure 2. Packets are funneled from nodes of higher rank to one of lower rank.

However, Figure 3 shows that a malicious node advertises the root rank even though it is not root.

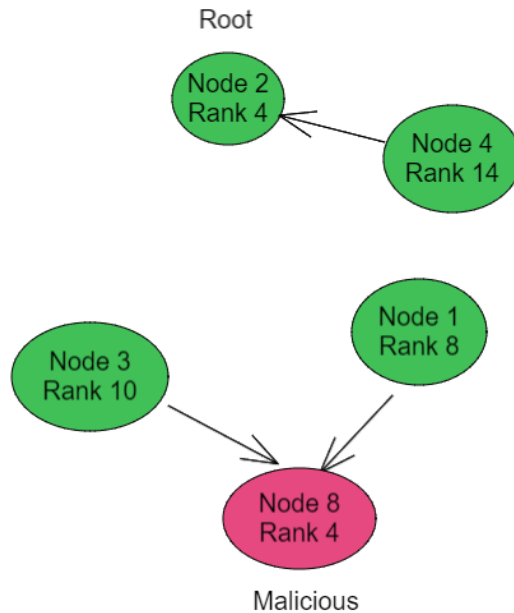


Figure 3. Malicious node falsely advertises a lower rank.

In this case, the malicious node with low rank receives packets from nearby nodes, however some nodes will still successfully deliver packets to the correct destination.

Since the RPL protocol works by creating optimal path based on rank information, when a benign node sends packet to a malicious node, it will calculate its own rank wrong, and therefore advertise incorrect route to other benign nodes, causing a sort of chain reaction. This attack is not extremely harmful, because the malicious node will send the interfered packet to a different node in the network and eventually reach the destination node (Packets in the RPL path will keep moving towards destination).

The real malicious behavior is achieved if the sinkhole is then used to do some sort of packet manipulation. In our case we use packet dropping to cause a real issue. The code modification is simple in this case, the packet handler method in the malicious node's code simply drops any received packet.

Images from running the simulation:

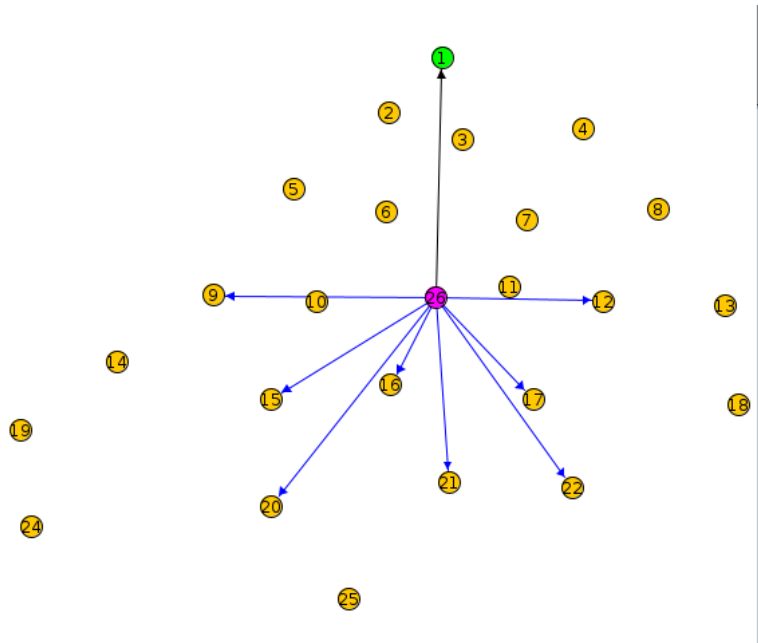


Figure 4. Node 1 is the Server to which the yellow nodes (regular Clients) will send data to. Node 26 is the attacking client node.

61695	ID:1	Got collect packet from node 13
68524	ID:1	Got collect packet from node 11
83914	ID:1	Got collect packet from node 7
87761	ID:1	Got collect packet from node 14
87793	ID:1	Got collect packet from node 8
88628	ID:1	Got collect packet from node 26

Figure 5. Node 26 is the sinkhole node advertising false rank (rank stolen from node 1).

The root node (ID:1) is receiving some full packets and some are only control packets

62373	ID:26	Dropping package from aaaa::212:7416:16:1616 heading to aaaa::1
72751	ID:26	Dropping package from aaaa::212:7417:17:1717 heading to aaaa::1
81250	ID:26	Dropping package from aaaa::212:7419:19:1919 heading to aaaa::1
82125	ID:26	Dropping package from aaaa::212:7411:11:1111 heading to aaaa::1
83873	ID:26	Dropping package from aaaa::212:7410:10:1010 heading to aaaa::1
87249	ID:26	Dropping package from aaaa::212:7418:18:1818 heading to aaaa::1

Figure 6. Node 26 is dropping packets from neighboring nodes. The addresses correspond to Nodes 22,23,25,17,16 and 24 respectively.

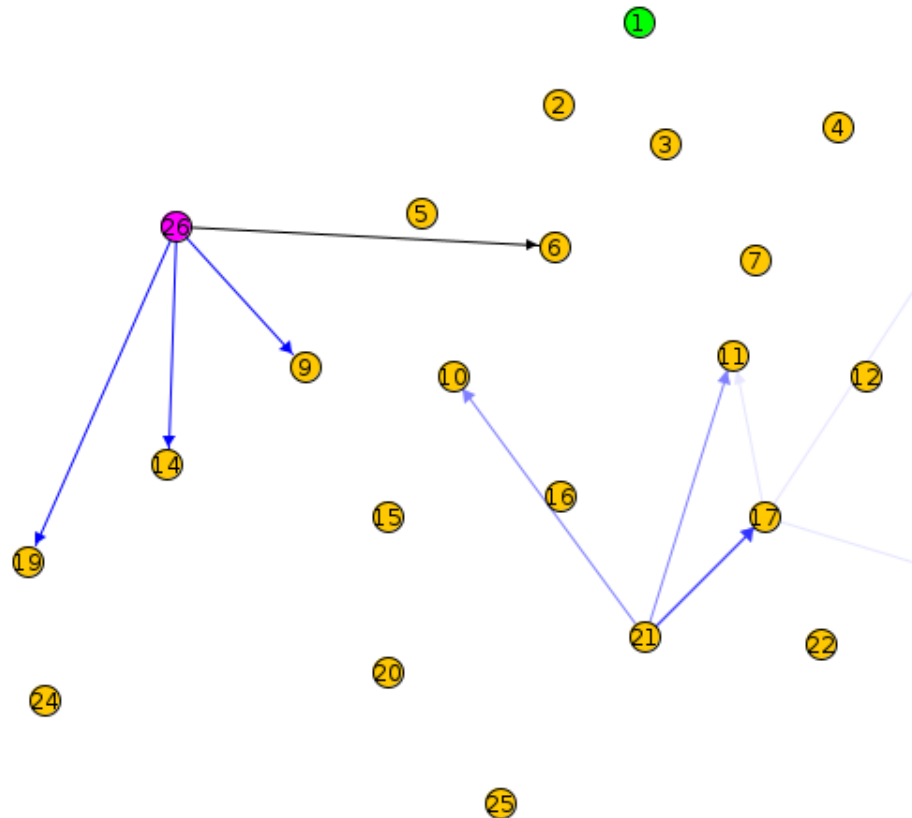


Figure 7. The location of the malicious node (26) has been changed and it advertises its rank based on a different root node (node with lowest rank) within its transmission range.

Black arrow showing local root node, Blue arrows showing nodes it receives packets from (fooled nodes)

In this case, it will drop more packets from 19,14,9 but will also drop packets from other nearby nodes if it does receive them. However most likely the other nodes will have a better path to send packet rather than node 26.

Defense

Cooja does not have a native Intrusion Detection (IDS), and after researching implementation methods for detecting sinkhole attacks it was advised to use some sort of network mapping to periodically check the advertised rank of nodes and perform a self-calculation using the routing table. We tried some basic implementation but we could not get all the data from the nodes to get it calculating other node's ranks. And since we were running out of time, we had to utilize a GitHub user's implementation (this was from 2012) of an IDS as the defense mechanism for this RPL Sinkhole attack and make some modifications so that it works in our Contiki systems. In general, an IDS is a monitoring system that monitors network operations and reports back relevant data when it detects a malicious node on the network. In our simulation, as packets were sent throughout the network, the node with the IDS installed was able to detect rank inconsistencies between what the malicious node was advertising and what it was supposed to be (rank calculation based on neighbor node information). The IDS then reported the malicious activity in Cooja's output window.

The IDS implementation is similar to what we were taught in the lectures; it performs a network mapping based on route table and RPL control messages from neighboring nodes. The client nodes look at the neighbors and collect their advertised rank information, which gets passed up the DODAG to reach the destination node (Root node of the entire network). All the other nodes perform the same activity. The IDS at the root node then compare the received information and check if the nodes within its transmission range are advertising the calculated rank. If not, report it as a suspected attacker.

```

240716 ID:1 Network graph at timestamp 2:
240717 ID:1
240738 ID:1 aaaa::212:7401:1:101 (t: 2, p: 0, r: 256) {}
240770 ID:1 aaaa::212:740b:b:b0b (t: 2, p: 0, r: 512)
240798 ID:1 aaaa::212:7405:5:505 (t: 2, p: 0, r: 512)
240831 ID:1 aaaa::212:740c:c:c0c (t: 2, p: 0, r: 512)
240859 ID:1 aaaa::212:740d:d:d0d (t: 2, p: 0, r: 768)
240891 ID:1 aaaa::212:7411:11:1111 (t: 2, p: 0, r: 784)
240919 ID:1 aaaa::212:7408:8:808 (t: 2, p: 0, r: 512)
240946 ID:1 aaaa::212:7402:2:202 (t: 2, p: 0, r: 512)
240977 ID:1 aaaa::212:740a:a:a0a (t: 2, p: 1, r: 768)
241002 ID:1 aaaa::212:740e:e:e0e (t: 2, p: 0, r: 768)
241020 ID:1 aaaa::212:7417:17:1717 (t: 2, p: 0, r: 1040)
241053 ID:1 aaaa::212:7416:16:1616 (t: 2, p: 0, r: 768)
241056 ID:1 aaaa::212:7417:17:1717
241072 ID:1 aaaa::212:7413:13:1313 (t: 2, p: 0, r: 1040)
241073 ID:1 -----

```

Network graph report from IDS with timestamp, ipv6 address and rank values

Process

To start the project, we looked at some papers that discussed attack and defense scenarios in RPL networks. The sinkhole attack was the most common method but also a fairly strong form of attack, we chose to go with that. Defense recommendations mentioned IDS very often and sometimes also firewalls, but implementing firewall required lot of changes to the protocol source code so we were looking into how an IDS for a sinkhole attack would work. As mentioned earlier our efforts were getting us nowhere so we had to use a pre-made IDS. After implementing the sinkhole attack with packet dropping, some experiments were done to see if the attack actually worked. Similarly for the IDS, we had to make some changes in Contiki file structure and modifying imports/makefiles for it to work since the IDS has no documentation and is from 2012. The experiment data collection was once again difficult because even pre-made examples by Contiki creators wouldn't work (and in the new version of Contiki, the analysis tool is completely removed). So, we had to parse log files to plot analyze the data.

Experimental Results

Since the attack method primarily focuses on rerouting packets to an attacker(s) node and then dropping them, we will see the effect on packets received by the server and packets dropped by the attacker. The server mote outputs "Packet received from node X" whereas the malicious mote outputs "packet drop from source X heading to destination Y". We will use this output to write to a file and read the results from there. The tool used to measure packet and transmission information (Collect-view tool) once again does not want to work. The issue is that the default implementation takes more ROM

than available in the device, so a simplified implementation was made which does not want to work (even though the information does reach the tool, it fails to unpack it). So, we will be doing a more manual collection of data.

The experiments we will perform on attacks are as follows: Network simulation with no attacker, simulation with attacker in 3 different locations in the network, simulation with 2 attackers (more coverage).

Simulation with no sinkhole:

With no sinkholes, we are expecting all the client nodes to send packets to the server (destination) node. The packets simply contain routing metric (rank information), number of hops taken, previous parent node and other default RPL information.

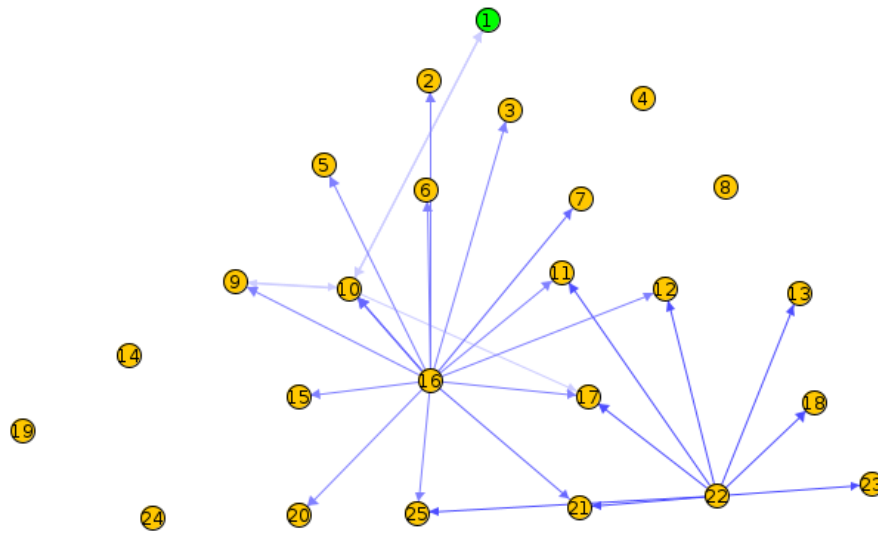
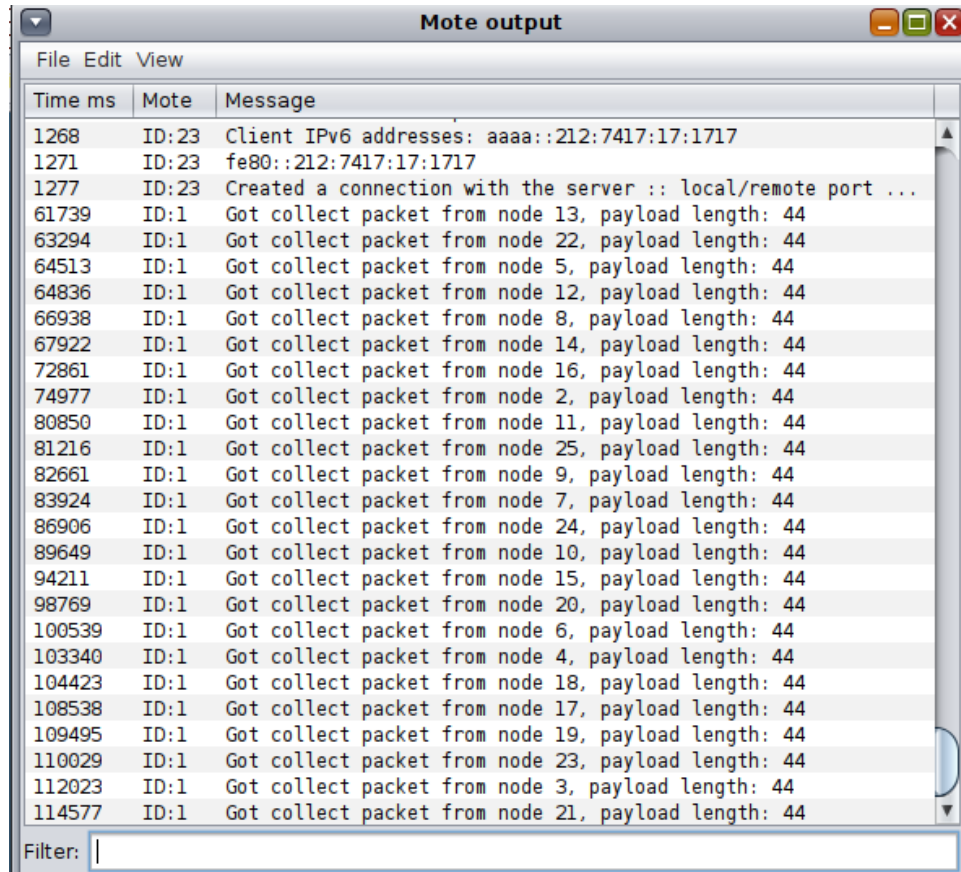


Figure 8. Normal network topology without sinkhole.



Time ms	Mote	Message
1268	ID:23	Client IPv6 addresses: aaaa::212:7417:17:1717
1271	ID:23	fe80::212:7417:17:1717
1277	ID:23	Created a connection with the server :: local/remote port ...
61739	ID:1	Got collect packet from node 13, payload length: 44
63294	ID:1	Got collect packet from node 22, payload length: 44
64513	ID:1	Got collect packet from node 5, payload length: 44
64836	ID:1	Got collect packet from node 12, payload length: 44
66938	ID:1	Got collect packet from node 8, payload length: 44
67922	ID:1	Got collect packet from node 14, payload length: 44
72861	ID:1	Got collect packet from node 16, payload length: 44
74977	ID:1	Got collect packet from node 2, payload length: 44
80850	ID:1	Got collect packet from node 11, payload length: 44
81216	ID:1	Got collect packet from node 25, payload length: 44
82661	ID:1	Got collect packet from node 9, payload length: 44
83924	ID:1	Got collect packet from node 7, payload length: 44
86906	ID:1	Got collect packet from node 24, payload length: 44
89649	ID:1	Got collect packet from node 10, payload length: 44
94211	ID:1	Got collect packet from node 15, payload length: 44
98769	ID:1	Got collect packet from node 20, payload length: 44
100539	ID:1	Got collect packet from node 6, payload length: 44
103340	ID:1	Got collect packet from node 4, payload length: 44
104423	ID:1	Got collect packet from node 18, payload length: 44
108538	ID:1	Got collect packet from node 17, payload length: 44
109495	ID:1	Got collect packet from node 19, payload length: 44
110029	ID:1	Got collect packet from node 23, payload length: 44
112023	ID:1	Got collect packet from node 3, payload length: 44
114577	ID:1	Got collect packet from node 21, payload length: 44

Figure 9. Normal network packets without sinkhole.

As expected, there are no packets dropped, and all client nodes have successfully delivered packets after simulation ran for 1 Minute 55 seconds.

Simulation with attacker position 1:

Next, we reset the simulation and place 1 attacker node on the left side of the network as shown in Figure 10.

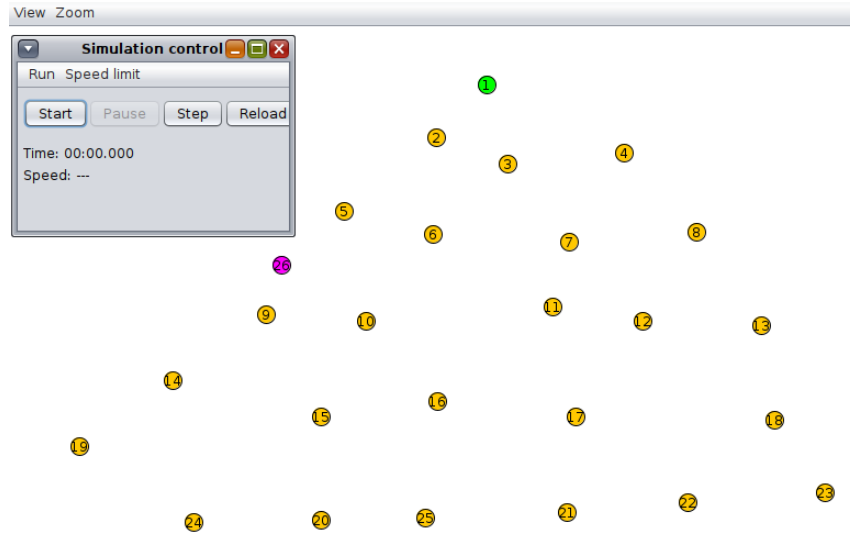


Figure 10. Malicious node 26 located on the left side of the network.

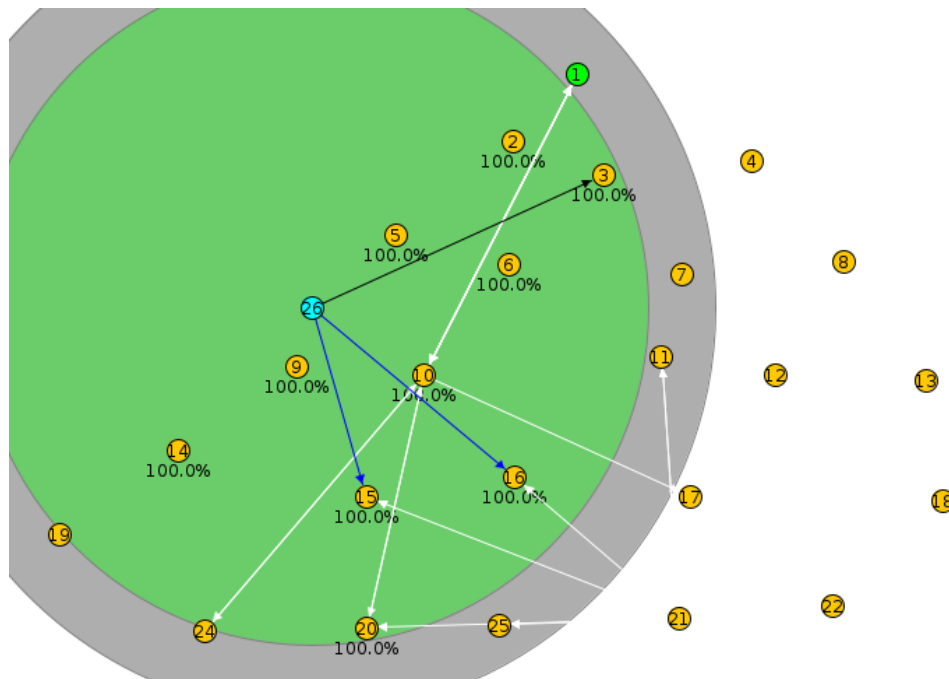


Figure 11. Network topology showing nodes within range of the malicious node.

We run the sim and wait for the nodes to initialize and discover neighbors. Here we see that Malicious node 26 has identified Node 3 as the root node (lowest rank) within its transmission range (green zone), so it advertises that rank as its own rank.

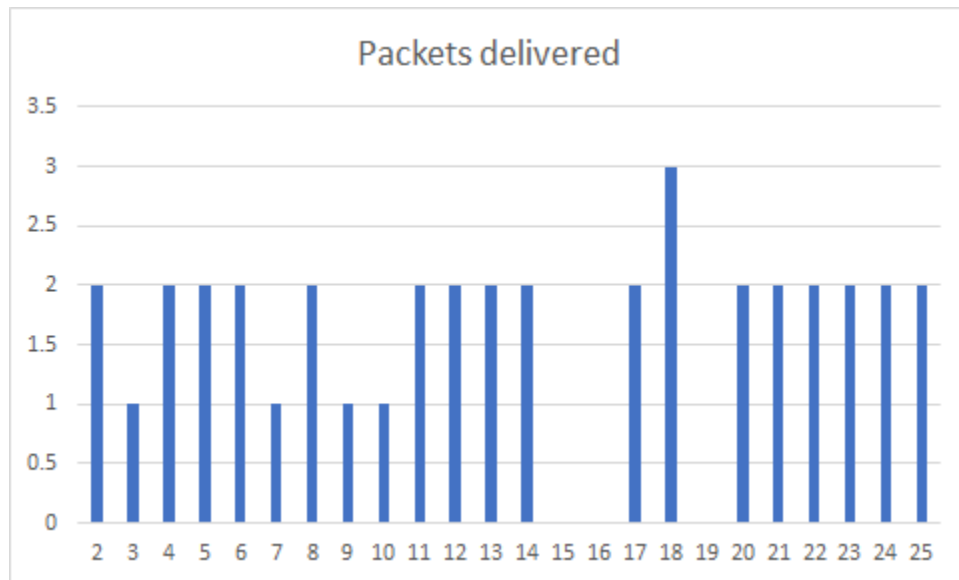
After running for 3 minutes, the sinkhole dropped 5 packets belonging to nodes 15,16,19:

7620	ID:26	#L 3 1
18495	ID:26	#L 15 1;blue
20495	ID:26	#L 15 1;blue
20618	ID:26	#L 16 1;blue
82874	ID:26	Dropping package from aaaa::212:740f:f:f0f heading to aaaa::1
83872	ID:26	Dropping package from aaaa::212:7410:10:1010 heading to aaaa::1
101248	ID:26	Dropping package from aaaa::212:7413:13:1313 heading to aaaa::1
141252	ID:26	Dropping package from aaaa::212:7413:13:1313 heading to aaaa::1
163372	ID:26	Dropping package from aaaa::212:7410:10:1010 heading to aaaa::1

Figure 12.

We check the output to see if server ever received from nodes 15,16,19, and if we filter the results, we see that no packets from these sources reached.

The graph below shows the number of packets received by server(Sink) from each node. (X-Axis shows node ID)



Graph 1.

Simulation with attacker position 2:

We now perform the same experiment but place the node to the right side of the network. Also run the simulation longer for better results.

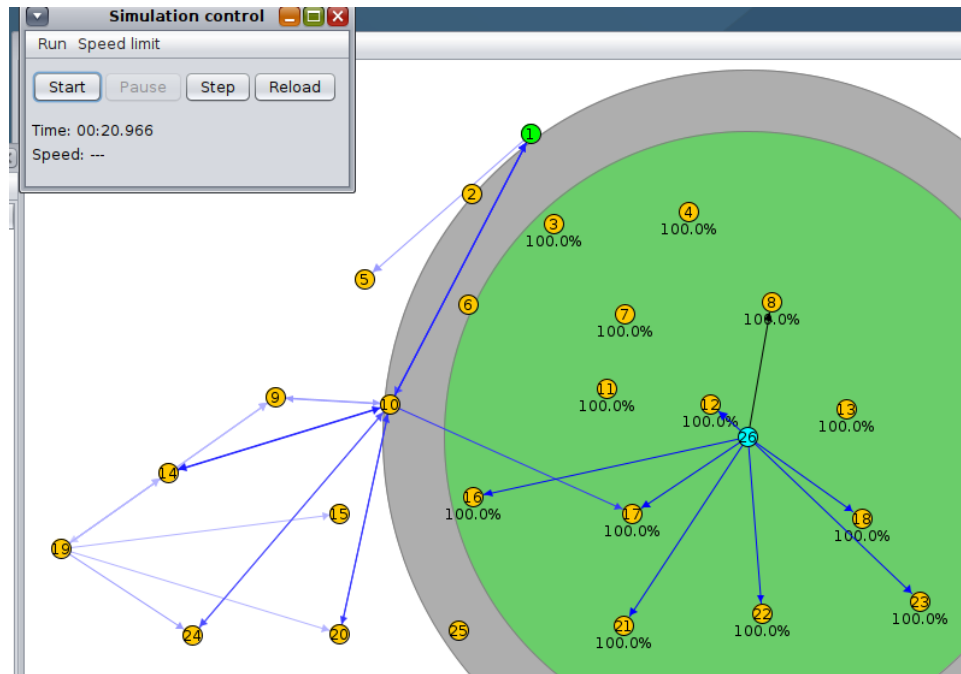
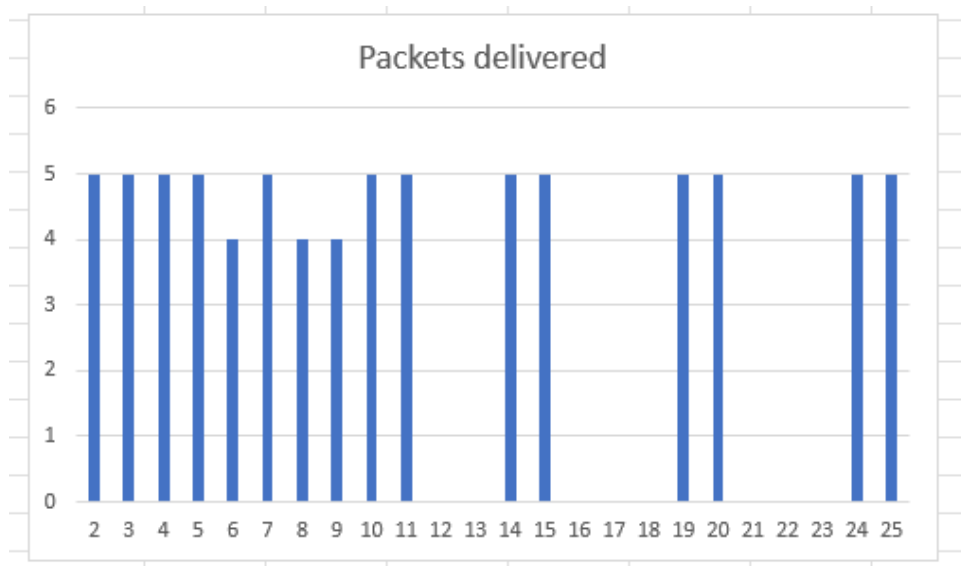


Figure 13.

This time, sinkhole node identifies Node 8 as the Root in the local DAG.

We ran the simulation for 6 minutes, and we saw more packets delivered but also a lot more packets dropped.



Graph 2.

Again, we see that on average, the nodes that are not affected by sinkhole, deliver 5 packets on average, Nodes 12, 13, 16, 17, 18, 21, 22 and 23 have no packets reaching the destination. If we look at the network diagram above, we see that the blue lines connecting node 26 are the affected nodes. This result makes sense because the nodes are not affected by sinkhole but are still within transmission

range and have found a better route up the DODAG network, so they avoid sending packets to Node 26 (sinkhole).

This position is also far more effective than Position 1, where it only affected 3 nodes.

Simulation with attacker position 3:

Figure 14 shows node 26 as the sinkhole located in the middle of the network to analyze if there is a greater effect on the packet delivery.

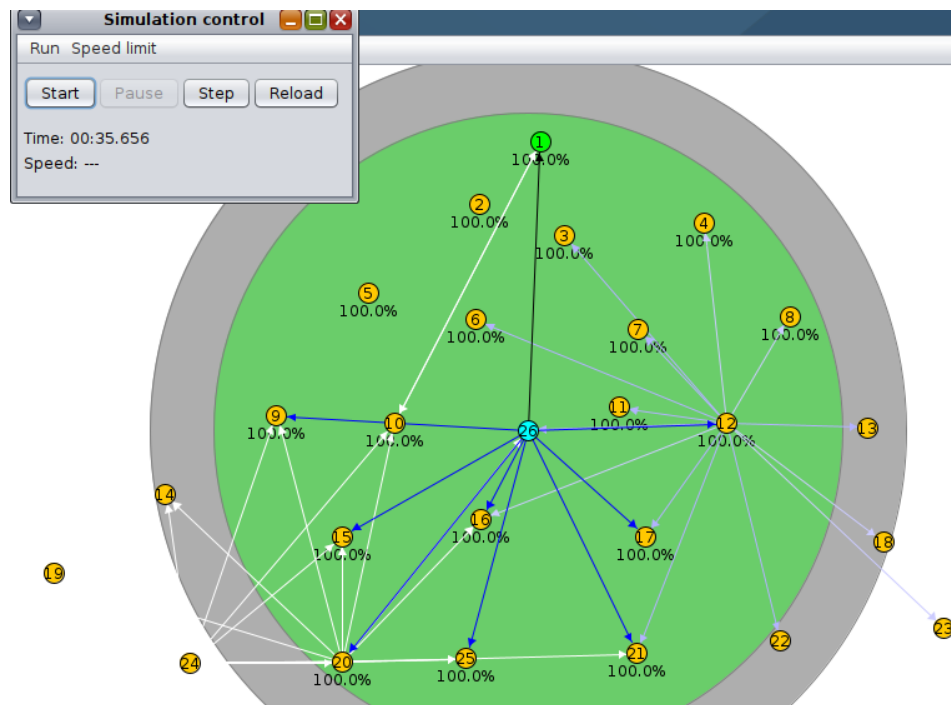
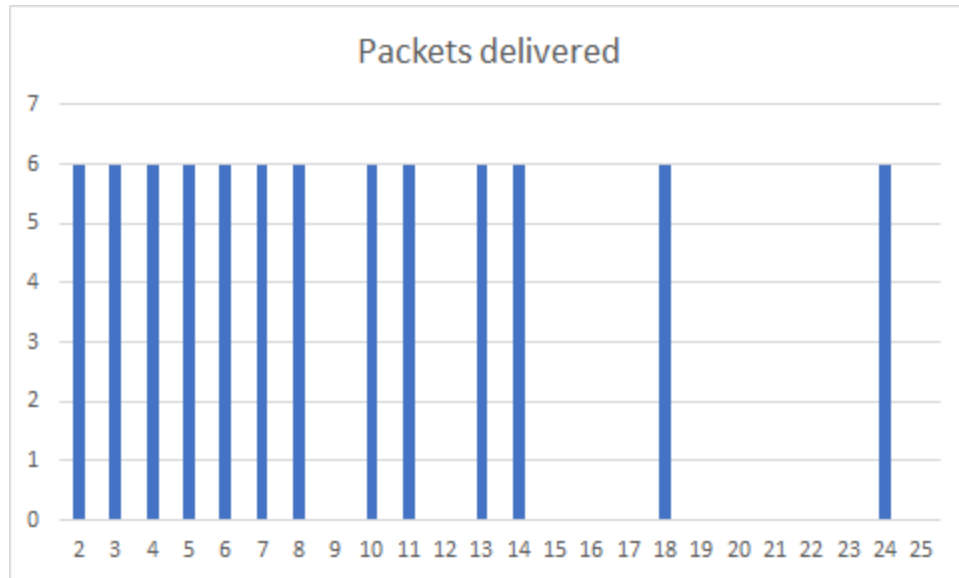


Figure 14. Malicious node 26 located in the center of the network topology.

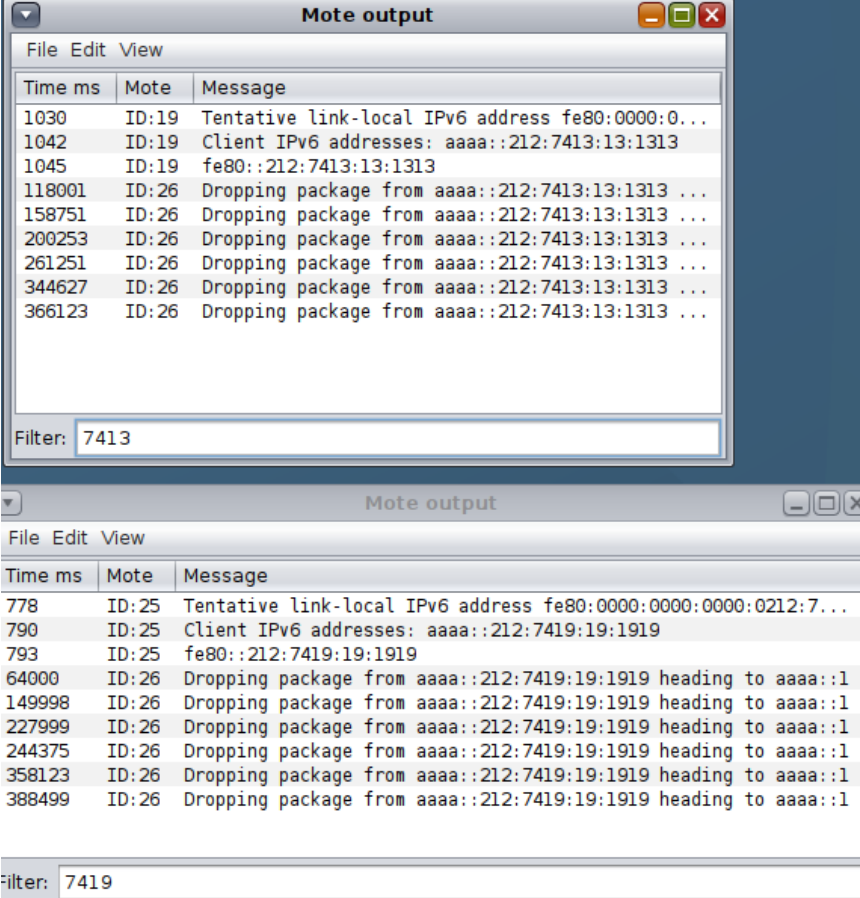
We can see that in this position the sinkhole has even better coverage. Also notice that since the destination node (ID: 1) is within transmission range, it uses that node's rank to advertise itself. We run the simulation for 6-7 minutes to check the output.



Graph 3.

Compared to Position 2, this central position affects 11 nodes and drops packets coming from nodes 9,12,15,16,17,19,20,21,22,23 and 25.

We can also confirm that the number of packets dropped by the sinkhole is 6 for each affected node. The image below shows 6 packets dropped from source nodes 19 and 25



Time ms	Mote	Message
1030	ID:19	Tentative link-local IPv6 address fe80:0000:0...
1042	ID:19	Client IPv6 addresses: aaaa::212:7413:13:1313
1045	ID:19	fe80::212:7413:13:1313
118001	ID:26	Dropping package from aaaa::212:7413:13:1313 ...
158751	ID:26	Dropping package from aaaa::212:7413:13:1313 ...
200253	ID:26	Dropping package from aaaa::212:7413:13:1313 ...
261251	ID:26	Dropping package from aaaa::212:7413:13:1313 ...
344627	ID:26	Dropping package from aaaa::212:7413:13:1313 ...
366123	ID:26	Dropping package from aaaa::212:7413:13:1313 ...

Filter: 7413

Time ms	Mote	Message
778	ID:25	Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7...
790	ID:25	Client IPv6 addresses: aaaa::212:7419:19:1919
793	ID:25	fe80::212:7419:19:1919
64000	ID:26	Dropping package from aaaa::212:7419:19:1919 heading to aaaa::1
149998	ID:26	Dropping package from aaaa::212:7419:19:1919 heading to aaaa::1
227999	ID:26	Dropping package from aaaa::212:7419:19:1919 heading to aaaa::1
244375	ID:26	Dropping package from aaaa::212:7419:19:1919 heading to aaaa::1
358123	ID:26	Dropping package from aaaa::212:7419:19:1919 heading to aaaa::1
388499	ID:26	Dropping package from aaaa::212:7419:19:1919 heading to aaaa::1

Filter: 7419

Figure 15.

In total, 65 packets were dropped and

Simulation with 3 attackers (full coverage):

Figure 16 shows the addition of 2 more attackers placed in the network to ensure full coverage and check how many packets deliver successfully. Nodes 26, 27, 28 are sinkholes.

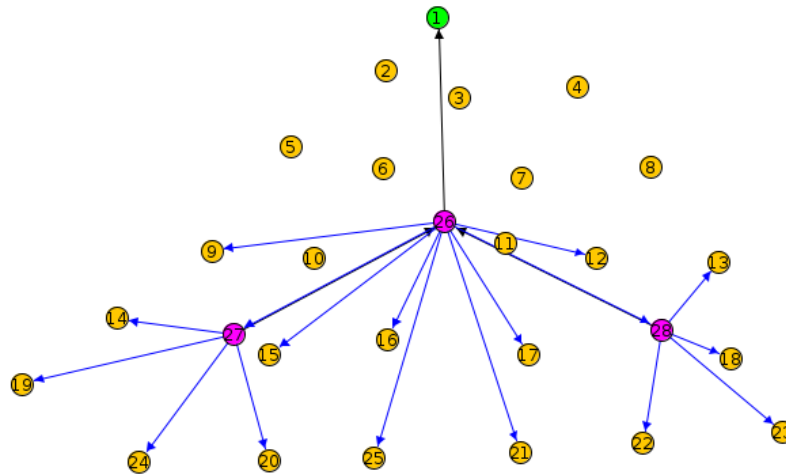


Figure 16. Three total malicious nodes in their original network locations.

Another thing to note from the network image is that node 26 advertises the rank from node 1, and now that node 26 has the best rank, the other sinkholes (27 and 28) use node 26's rank as their advertising rank. After realizing that this may not be best for maximum coverage, we change their positions to prevent this overlapping as can be seen in Figure 17.

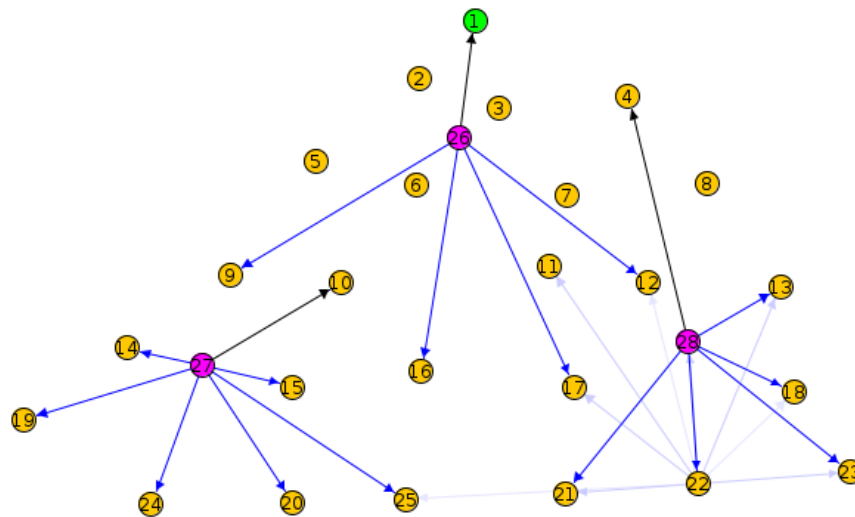
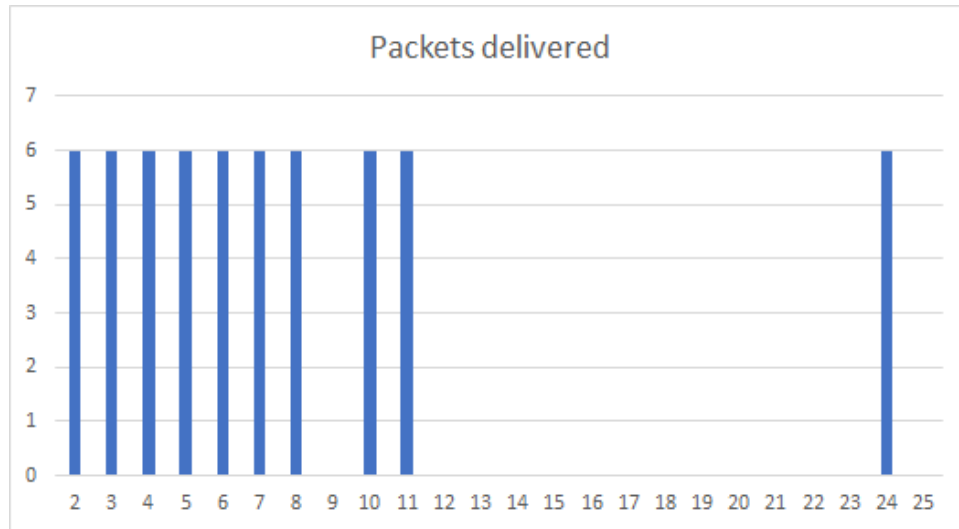


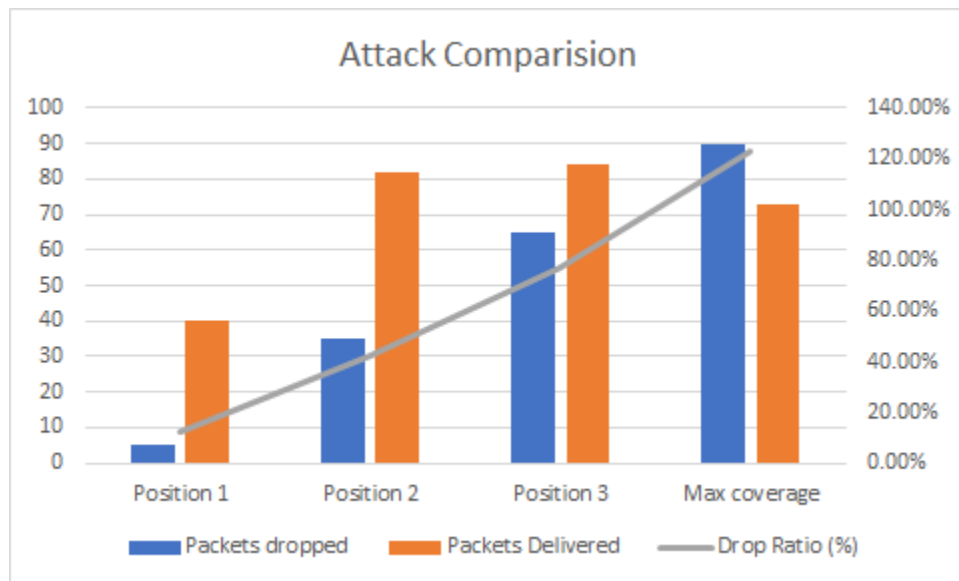
Figure 17. The locations of the three attacker nodes spread out on the network.



Graph 4.

Now we see that most of the nodes are unable to deliver packets to destination Node 1.

From these 4 experiments, we can plot the number of packets dropped and delivered and see their ratio. See Graph 5.



Graph 5. Packets delivered and dropped for multiple node locations.

This chart indicates that with 1 sinkhole present in the network, Position 3; a central position is the best to drop the maximum number of packets. When we add multiple sinkholes, there are more packets dropped than delivered. But hypothetically there will be a diminishing advantage as more sinkhole nodes are added to the network, making them easier to detect.

Sinkhole position	Packets dropped	Packets Delivered	Drop Ratio (%)
Position 1	5	40	12.50%
Position 2	35	82	42.70%
Position 3	65	84	77.40%
Max coverage	90	73	123.30%

Table 1.

The experiments we perform on the defense are as follows: Check if IDS identifies that there are no issues in the network with 0 attackers, identify 1 attacker correctly, identify multiple attackers correctly.

IDS with no attacker (Checking for True Negative):

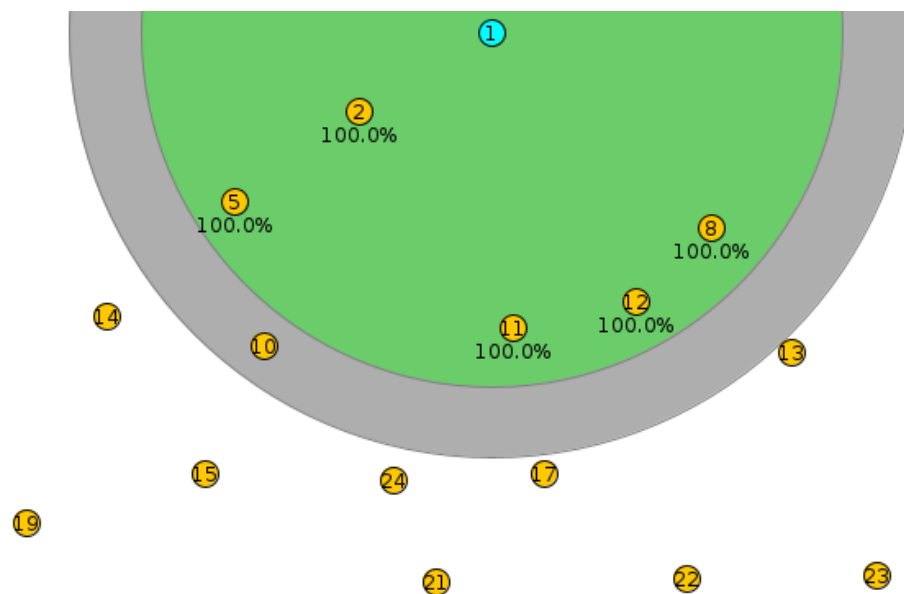


Figure 18. Network with IDS and no sinkhole

In these experiments, Node 1 is the sink, and maps the network based on routing information it receives from its neighbors. The IDS functionality is built into this node.

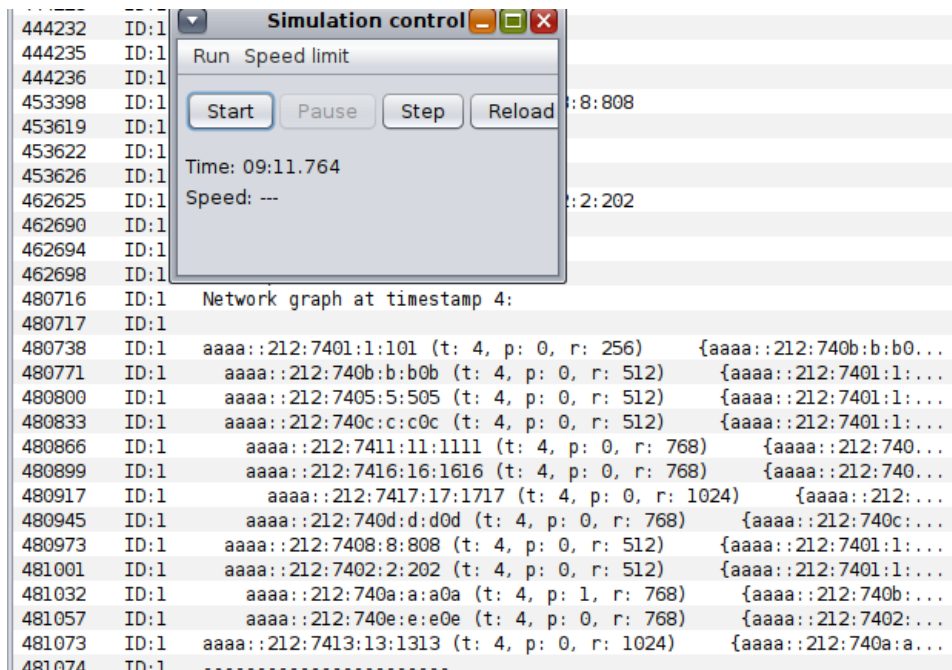


Figure 19. Reported network graph by IDS, no attackers to report

After running for around 9 minutes, the IDS finds nothing wrong in the network and continues to update the network mapping. The r values are the routing_metric or rank values of the nodes assigned to that ipv6 address.

IDS with 1 attacker:

We place an attacker node (Node ID:25) in the network and place it just outside transmission range on the IDS, to check if it can detect it.

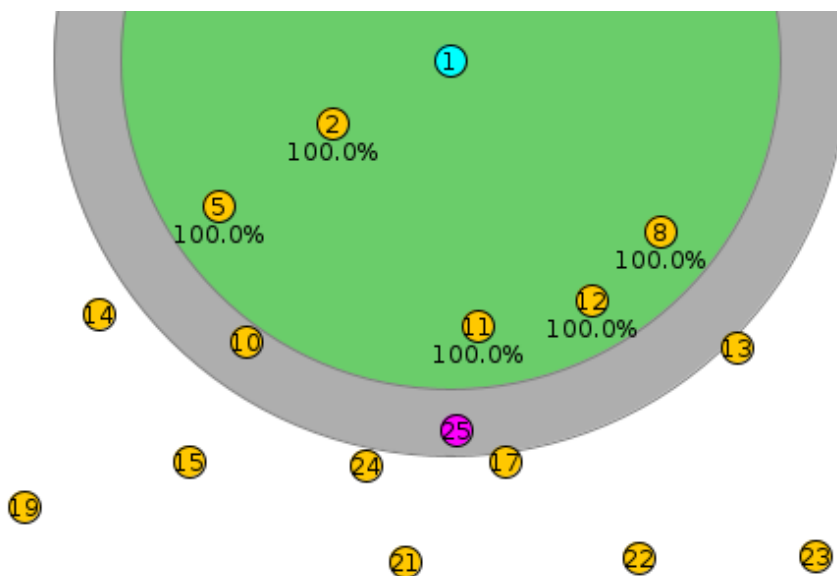


Figure 20. Network with IDS and 1 sinkhole outside range

At this position it failed to detect the sinkhole.

So, we move the node into transmission range and it is expected to work now.

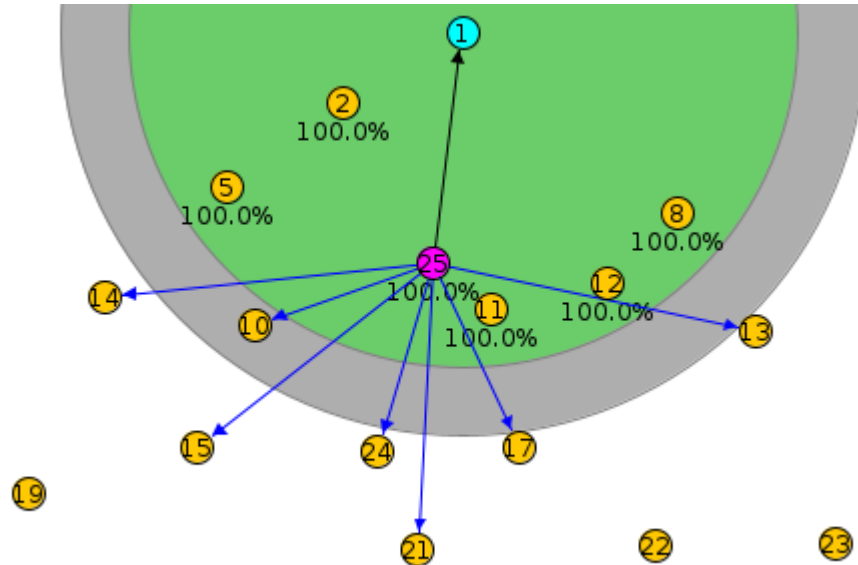


Figure 21. Network with IDS and 1 sinkhole inside TX range

Now that Node 25 is within range, the IDS detects that something is interfering with the network as it reports that some nodes have outdated or non-existent routing information:

```
240952 ID:1 The following list of nodes either have outdated or non-existent infor...
240955 ID:1 aaaa::212:7413:13:1313
240958 ID:1 aaaa::212:7417:17:1717
240962 ID:1 aaaa::212:7415:15:1515
240964 ID:1 aaaa::212:740a:a:a0a
240968 ID:1 aaaa::212:7411:11:1111
240970 ID:1 aaaa::212:740f:f:f0f
```

Figure 22. List of nodes with outdated RPL information

These addresses correspond to node IDs: 23,19,21,10,17 and 15 respectively. The reason that it mentions these addresses before it even detects the sinkhole node is probably because the sinkhole is preventing RPL control packets from reaching the Sink node (Node 1).

But after a few minutes it does detect that Node 25 is advertising the wrong rank:

```
1119 ID:25 Tentative link-local IPv6 address fe80:0000:0000
1132 ID:25 Client IPv6 addresses: aaaa::212:7419:19:1919
-----
480948 ID:1 The following nodes has advertised incorrect routes:
480952 ID:1 aaaa::212:7401:1:101 (256)
480955 ID:1 aaaa::212:7419:19:1919 (256)
```

Figure 23. IDS reporting node with false rank/route

The address 7419 is of Node 25, the other address it detects as incorrect is itself (Node 1). Since Node 25 copies and advertises Node 1's rank, it detects something wrong with those 2 nodes.

The issue with an IDS is that it detects an attack, but cannot stop the attack on its own. While the IDS was detecting, the Sinkhole was busy dropping packets:

268744	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:740a:a...
277992	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:7411:1...
287243	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:740f:f...
361119	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:7413:1...
370367	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:7417:1...
379494	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:7415:1...
388744	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:740a:a...
397992	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:7411:1...
416867	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:740f:f...
481120	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:7413:1...
490368	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:7417:1...
499494	ID: 25	Dropping package from aaaa::212:7401:1:101 heading to aaaa::212:7415:1...

Figure 24. Packets dropped by sinkhole

IDS with 3 attackers (the same positions as maximum coverage experiment):

In this experiment, we add more sinkholes, 1 is within TX range and the 2 others are not. In this configuration we don't expect IDS to find sinkholes 26 and 27.

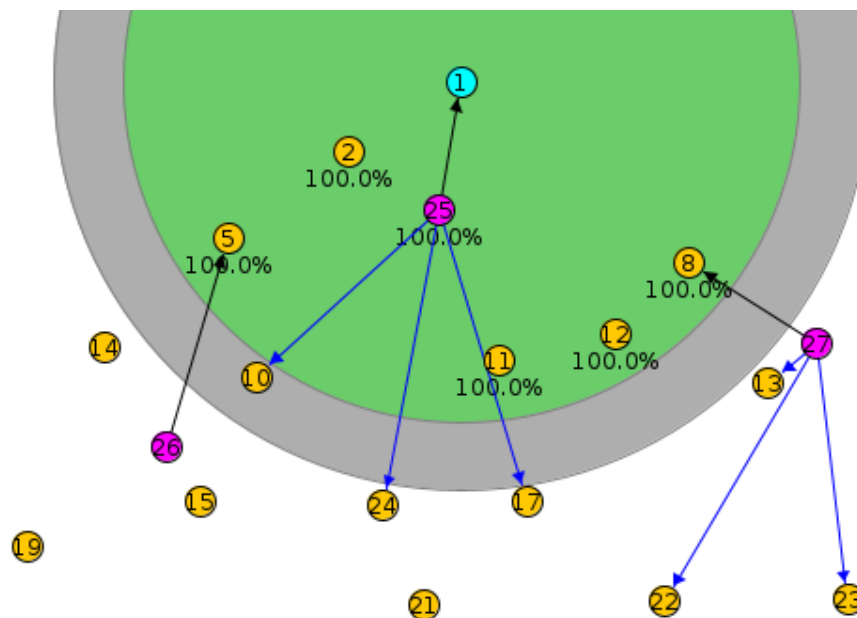


Figure 25. Network with IDS and 3 sinkholes (2 out of range)

After simulating 9 minutes, the IDS detects some fake rank advertising nodes:

```

480985 ID:1 The following nodes has advertised incorrect routes:
480988 ID:1 aaaa::212:7401:1:101 (256)
480992 ID:1 aaaa::212:7405:5:505 (512)
480996 ID:1 aaaa::212:7419:19:1919 (256)
481000 ID:1 aaaa::212:741a:1a:1a1a (256)
481005 ID:1 The following list of nodes either have outdated or non-existe

```

Figure 26. IDS reporting 3 sinkhole nodes

These addresses (excluding 1st one, it is self-address) correspond to nodes 5, 25 and 26.

So, it correctly detects 25 and 26, false positive detection of node 5 and fails to detect 27. The IDS reported the same even after simulating an additional 10 minutes.

Now we also wanted to experiment and see what would happen if the TX range of IDS was maximized to cover all the nodes in the network. Surely it must detect all the sinkholes now, so lets see.

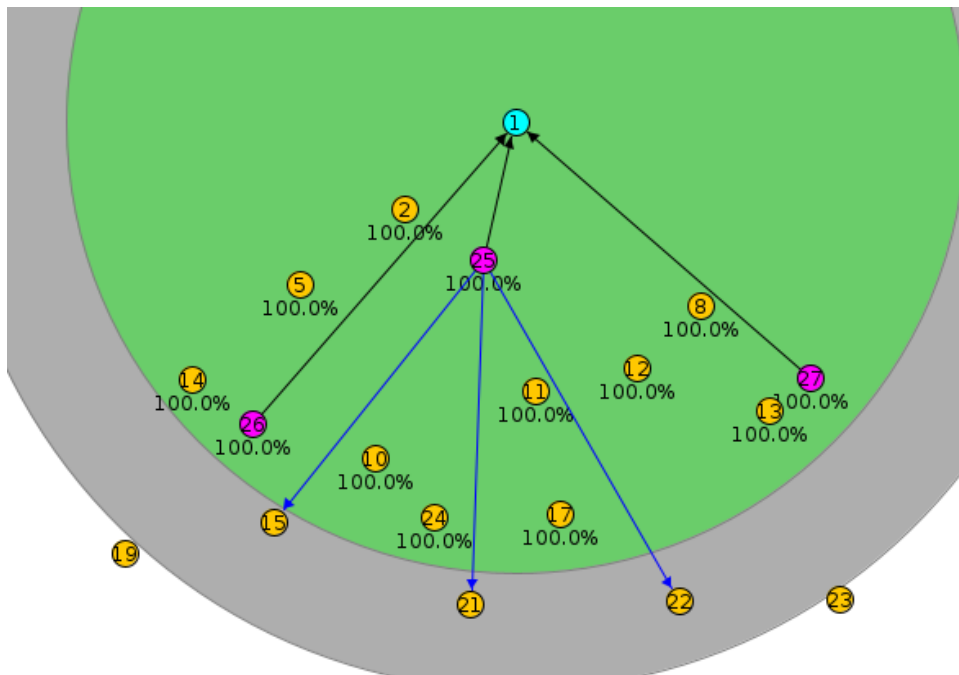


Figure 27. Network with increased transmission range

The simulator doesn't allow changing TX range for individual nodes, so it increases for all nodes. This means that all 3 sinkholes will identify the local Root node as node 1 and tries to steal its rank.

We see some output detecting rank inconsistencies:

```

481091 ID:1 Node 26 is claiming node 14 has rank 1536, while it claims it has 512
481096 ID:1 Node 26 is claiming node 24 has rank 1536, while it claims it has 512
481101 ID:1 The following nodes has advertised incorrect routes:
481104 ID:1 aaaa::212:7419:19:1919 (256)
481108 ID:1 aaaa::212:741b:1b:1b1b (256)
481112 ID:1 aaaa::212:741a:1a:1a1a (256)
481116 ID:1 The following list of nodes either have outdated or non-existe

```

Figure 26. IDS reporting 3 sinkhole nodes (correctly this time)

The addresses 7419, 741b, 741a correspond to nodes 25,26 and 27. So it does detect all 3 sinkholes as expected.

The conclusion to these tests is that for the IDS to work, the attacking nodes must within transmission range to detect malicious rank advertisement, and we also found that sometimes it falsely detects a benign node as a sinkhole (only if that benign node is a local Root to one of the sinkhole nodes).

Challenges Addressed

It was a challenge at first to get the simulation running. Finding out what code handled the rank and knowing what code to alter and execute was a challenge. Once we got the simulation running, it was fairly easy to see everything that was happening. However, we spent a lot of time trying to get the attack and defense working, so originally we wanted dynamic nodes using Mobility tool, but didn't have time to implement it in the new simulations. (Also we thought it would not be very helpful because the mobility plugin makes the nodes move too fast. So, the packet transfer speed and movement speed are not scaled properly, and the experiments would have more variables than just the attack vector).

In future network related experiments, we will look into other standalone network simulators because the main problem with using Cooja in Contiki is that it is very barebones (as expected for IoT), but this makes it very difficult to collect Node/network statistics since the implementation is very complicated and is out of our scope to get it working within a semester.

Some other challenges we noticed with Contiki is how they compartmentalize the source code for the protocols and other OS functionality. They are very different in the newer Contiki-NG compared to previous numbered version releases. For instance, the built-in example codes wouldn't work due to this file structure mismatch. Maybe it is better for the future, but it makes it quite difficult to look at older resources and use them for reference (many OS files are renamed and some are omitted and mixed with other source code, etc). So we end up spending more time understanding the Contiki architecture and less time implementing our own defense measure.

Team Member Contributions

This project was a group effort from the beginning since we all had an interest in vehicular IoT. From there we decided to explore the sinkhole attack and Intrusion Detection Systems as they are valid considerations for vehicular IoT networks. Tia took the lead on the research with assistance from Loken looking at papers on sinkhole attacks and vehicular IoT applications. Abir contributed by setting up Cooja for a sinkhole attack with the IDS. He analyzed the code and was able to get a working simulation. Loken assisted with the simulation as well and performed concurrent simulations on a separate machine. The project paper was a group effort to complete and compile the project findings.

References

List of references used in the project.

<https://core.ac.uk/download/pdf/324199534.pdf>

<https://github.com/ecksun/Contiki-IDS/>

<https://ieeexplore.ieee.org/document/7087034>