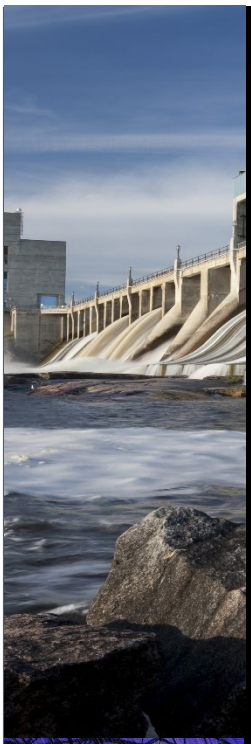


Rev: 24th April 2023

Using GTNET and NS-3 for Cyber-Physical Simulation

RTDS Technologies

Contents

1 CASE INFORMATION

Case Name	CyberSecSimPeakShaveUDP.rtfx
Location	\Example Cases\11 Cybersecurity\NS-3 peak shave\CyberSecSimPeakShaveUDP.rtfx
Created by	Chamara Devanarayana
Revision	00 – April 2023
Target	Cyber-Physical simulations using NS-3
Minimum Hardware	1 x NovaCor Chassis with at least 2 enabled cores 1 x GTNETx2
RSCAD Version	RSCAD FX 2.0 and above
Keywords	Cyber-security, GTNET-SKT
Purpose	Demonstrate a Man-In-The-Middle attack and a DoS attack on an energy market application using the GTNET-SKT UDP protocol.

Case Name	CyberSecSimPeakShaveTCP.rtfx
Location	\Example Cases\11 Cybersecurity\NS-3 peak shave\CyberSecSimPeakShaveTCP.rtfx
Created by	Chamara Devanarayana
00 – April 2023	00 – April 2023
Target	Cyber-Physical simulations using NS-3
Minimum Hardware	1 x NovaCor Chassis with at least 1 enabled core 1 x GTNETx2
RSCAD Version	RSCAD FX 2.0 and above
Keywords	Cyber-security, GTNET-SKT

Purpose	Demonstrate a Man-In-The-Middle attack and a DoS attack on an energy market application using the GTNET-SKT TCP protocol.

Case Name	GTNET_DNP.rtfx
Location	Tutorial Cases\03 Protection and Automation\06 GTNET Applications\06 SCADA\06a GTNET_DNP\GTNET_DNP.rtfx
Created by	Dinesh Gurusinghe
Revision	18 – June 2021
Target	Cyber-Physical simulations using NS-3
Minimum Hardware	1 x NovaCor Chassis with at least 1 enabled core 1 x GTNETx2
RSCAD Version	RSCAD FX 2.0 and above
Keywords	Cyber-security, GTNET-DNP
Purpose	Demonstrate a Man-In-The-Middle attack on DNP3 SCADA application.

Case Name	Modbus_Communication.rtfx
Location	\Tutorial Cases\03 Protection and Automation\06 GTNET Applications\06 SCADA\06c GTNET_MODBUS\Modbus_Communication.rtfx
Created by	Sachintha Kariyawasam
Revision	20 – September 2022
Target	Cyber-Physical simulations using NS-3
Minimum Hardware	1 x NovaCor Chassis with at least 1 enabled core 1 x GTNETx2

RSCAD Version	RSCAD FX 2.0 and above
Keywords	Cyber-security, GTNET-MODBUS
Purpose	Demonstrate a Man-In-The-Middle attack on MODBUS SCADA application.

Case Name	IEC104_Communication.rtfx
Location	\Tutorial Cases\03 Protection and Automation\06 GTNET Applications\06 SCADA\06b GTNET_104\ IEC104_Communication.rtfx
Created by	Dinesh Gurusinghe
Revision	12 – December 2022
Target	Cyber-Physical simulations using NS-3
Minimum Hardware	1 x NovaCor Chassis with at least 1 enabled core 1 x GTNETx2
RSCAD Version	RSCAD FX 2.0 and above
Keywords	Cyber-security, GTNET-IEC104
Purpose	Demonstrate a Man-In-The-Middle attack on IEC104 SCADA application.

Case Name	PMU_Communication.rtfx
Location	\Tutorial Cases\03 Protection and Automation\06 GTNET Applications\05 Synchrophasors\05a GTNET_PMU\PMU_Communication.rtfx
Created by	Dinesh Gurusinghe

Revision	21 – August 2021
Target	Cyber-Physical simulations using NS-3
Minimum Hardware	1 x NovaCor Chassis with at least 1 enabled core 1 x GTNETx2
RSCAD Version	RSCAD FX 2.0 and above
Keywords	Cyber-security, GTNET-PMU
Purpose	Demonstrate a Man-In-The-Middle attack on Synchrophasors to report false data.

Case Name	GOOSE_Communication.rtfx
Location	\Example Cases\11 Cybersecurity\GTNET_GSE\GOOSE_Communication.rtfx
Created by	Dinesh Gurusinghe
Revision	20 – March 2023
Target	Cyber-Physical simulations using NS-3
Minimum Hardware	1 x NovaCor Chassis with at least 1 enabled core 1 x GTNETx2
RSCAD Version	RSCAD FX 2.0 and above
Keywords	Cyber-security, GTNET-GSE
Purpose	Demonstrate a Man-In-The-Middle attack on GOOSE

Case Name	SV_Communication.rtfx
Location	\Tutorial Cases\03 Protection and Automation\06 GTNET Applications\01 Sampled Values\01a GTNET_SV\SV_Communication.rtfx

Created by	Sachintha Kariyawasam
Revision	4 – August 2021
Target	Cyber-Physical simulations using NS-3
Minimum Hardware	1 x NovaCor Chassis with at least 1 enabled core 1 x GTNETx2
RSCAD Version	RSCAD FX 2.0 and above
Keywords	Cyber-security, GTNET-SV
Purpose	Demonstrate a Man-In-The-Middle attack on Sampled values protocol.

2 INTRODUCTION

The traditional electric grid consisted of generation plants, transformers, tripping devices and feeders. The generation was mostly centralized, and the control was manual. The emergence of smart grid enabled two-way communication between the controller and assets in real time. This made it possible to include intermittent renewable energy resources such as wind and solar energy. Furthermore, it also enabled secondary energy markets, where the customer are paid for the limited energy they generate and the time shifting of their power consumption. However, this connectivity of controllers and assets through computer networks created the possibility of cyber-attacks on the power systems. The readers can refer [1] for a detailed survey of the possible attacks. In this report, we demonstrate how to simulate MITM attack and a DoS attack in an open source network simulator called NS-3. Then we show how to monitor the effects of these attacks in the power system using the RTDS™ with the aid of an example scenario. In this scenario, the Distribution system operator (DSO) makes use of secondary energy markets to carry out peak shaving.

2.1. Example scenario

In this report we are using a scenario that was presented in [2] at the Cyber-physical security for low-voltage grids website.

The Distributed energy resources (DERs) are customers that provide flexibility to the Distribution system operator (DSO) by mean of time shifting the load or providing small amounts of power. However, it is not feasible for the DSO to contract these DERs directly. Therefore, they make use of an entity called the Aggregator. The aggregators contract a portfolio of DERs. These aggregators in turn have contracts with the DSO. Whenever the DSO needs to shed some load these aggregators are informed of the shedding needed. The

aggregator operates the technical infrastructure to communicate with the DER units. These aggregators are financially responsible to the DSO in case the level of load shedding could not be delivered.

In this report, we are simulating a smart grid where the peak load of the distribution transformer is controlled by means of load shedding or increased generation at the DERs. In this scenario, the communication network consists of the following network connections:

- A remote terminal unit (RTU) and the DSO.
- DSO and the Aggregator
- The Aggregator and the DERs.

The RTU communicates data on the load level at the distribution transformer to the DSO. This communication network is modelled in NS-3. We can also model it in DeterLab which is a testbed containing actual computers. The power network related to this scenario is modelled in the RTDS.

Similar co-simulation setups have been used in the recent literature. In [2], Liu et al. conducted a co-simulation using RTDS and NS-3. In this simulation they made the state information of an IEEE 14-bus system available using the phasor measurement unit (PMU) and based on the information they performed closed loop control actions. Then the effect of DoS attacks of different magnitudes and MITM attacks were observed. In [3], Chen et al. analyzed the effects of cyber-attacks on the power system transient stability of bus voltage which uses a static VAR compensator. Here, they use an 11-bus test system modelled in RTDS. The cyber network was modelled in OPNET. Hahn et al. in [4] analyzed the cyber-physical impacts of malicious breaker tripping at generators observing the synchronicity of generator rotor angle, DoS attacks on the DNP3 servers and coordinated cyber-attacks. Here, the power system is modelled in RTDS and the cyber system is modelled in the PowerCyber testbed of the Iowa state university.

The remainder of this report is organized as follows. First, we introduce our system model which consists of the power system and the cyber-network with brief explanations on the

attacks that we are running. Next we explain how to connect the RTDS simulator to NS-3. Then we show how to simulate the cyber-physical system using the NS-3 simulator to model the network and attacks and RTDS simulator to model the power system

2.2. Installing NS-3

Follow the steps stated below to install NS-3 in you Ubuntu Linux PC.

1. In an Ubuntu Linux PC clone the git repository https://github.com/chamara84/ns3_cybersec.git

2.2.1. Installing the pre-requisites:

1. Go to ./ns3_cybersec/ns-allinone-3.29
2. ./configurePreReq
3. export CXXFLAGS="-Wall"
4. ./build.py

2.3. Create the ns3 configuration file for Protocol data modification

1. Go to /etc : cd /etc
2. Create folder ns3: sudo mkdir ns3
3. Change permissions for your username: chown <username>:<username> ns3
4. Go to folder ns3
5. Move the file ns3.conf in the ns3_cybersec folder to /etc/ns3/ folder.

3 SYSTEM MODELLING

The scenario is modelled in two parts. The first part is the power system model in the RTDS. The second is the communication network model in the NS-3. We are providing the details of these two models in the next two subsections. The power system model is shown in Figure 1.

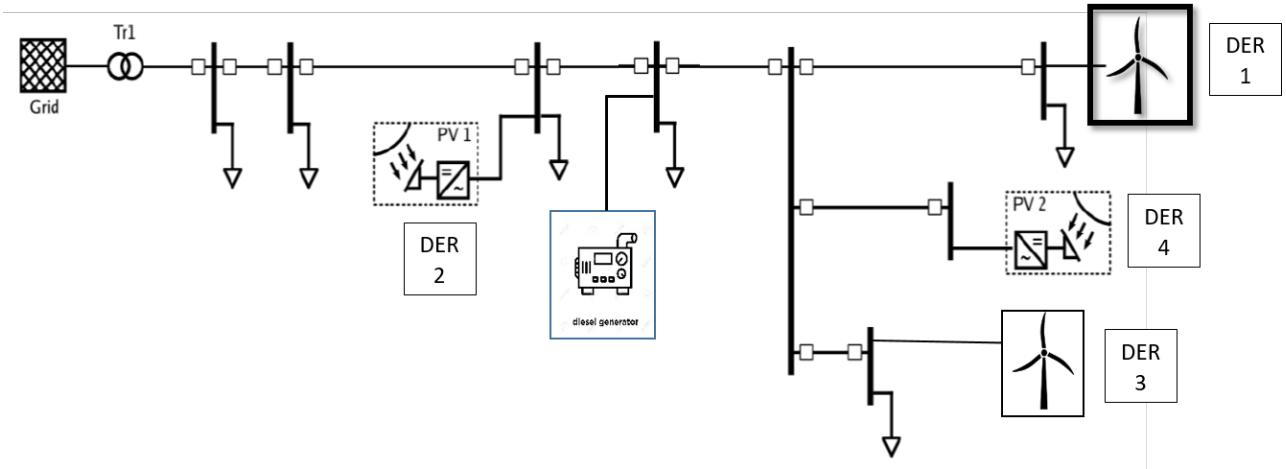


Figure 1 Power system model

3.1. RTDS simulation system model

Here, we are modelling a power system having four DERs. Each DER is represented with a Dynamic source and a Dynamic load. Then the grid transmission is modelled as a power source of 4kV. The Distribution transformer steps it down to 120V. The power factor was assumed to be 0.8 and we needed to keep the load of the distribution transformer at 6MVA. This load level is measured in real time using the P & Q meter model in the RSCAD.

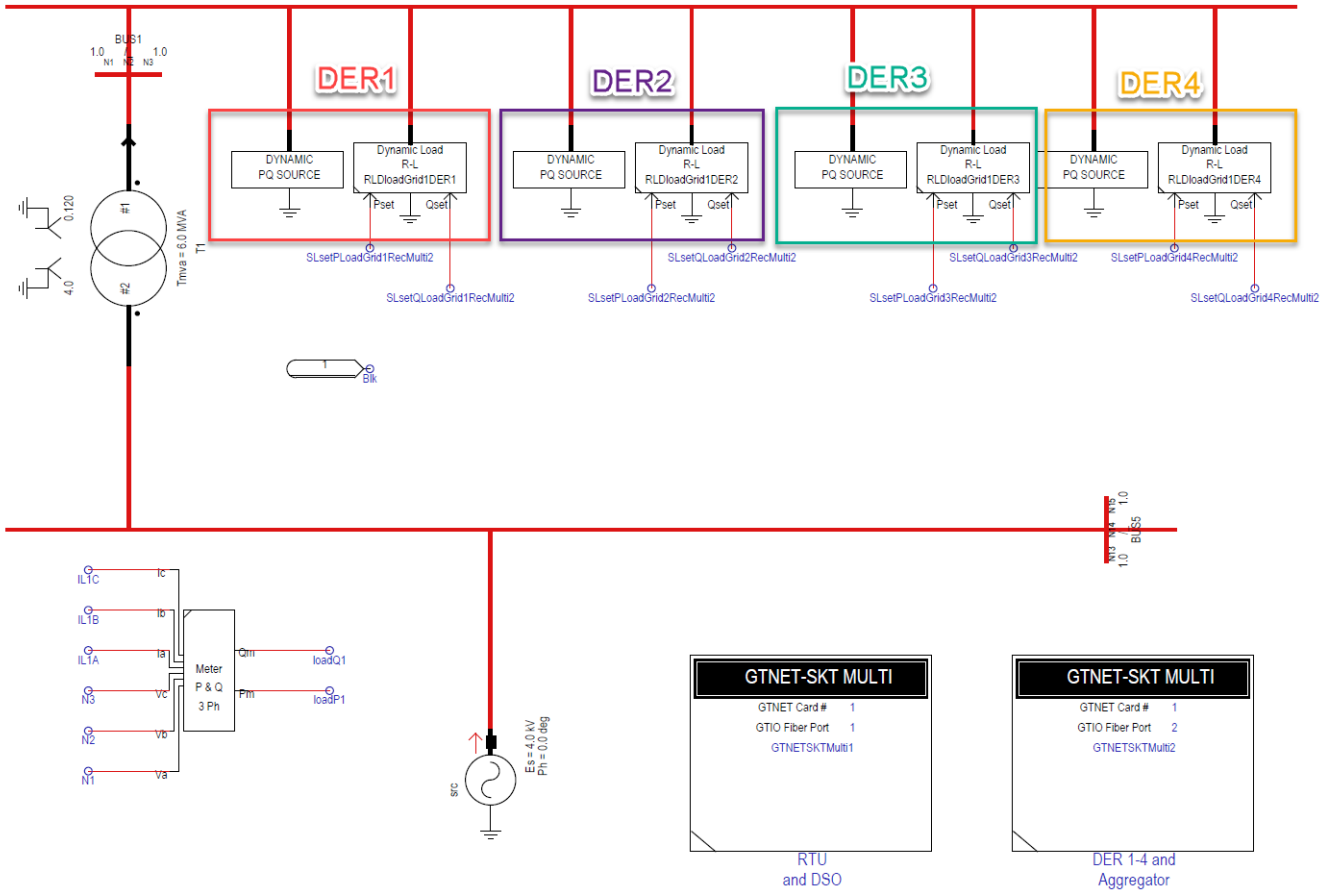


Figure 2 Power system simulation model

3.2. NS-3 Communication Network Simulation model

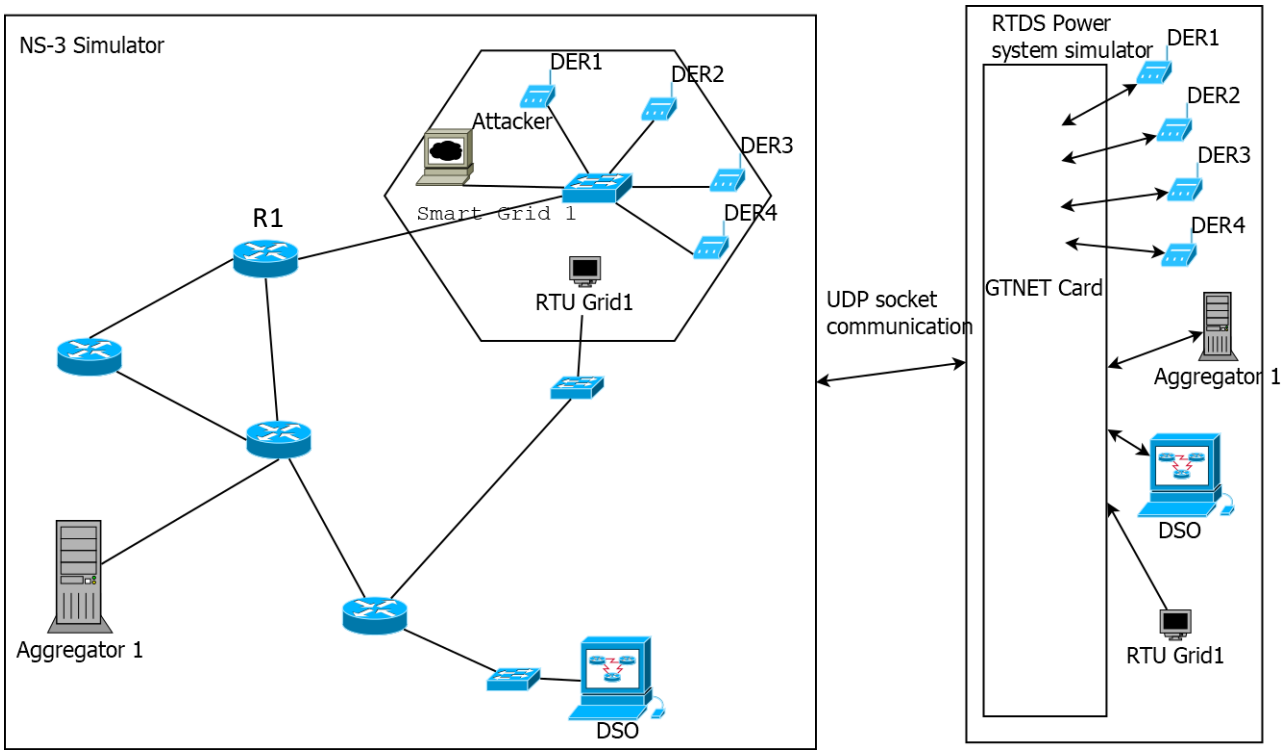


Figure 3 Network simulation model

We model all the network components using the Node type in the NS-3 network simulator. Then all the connections between nodes other than the DERs, the Attacker and the Switch in the Smart Grid1 are modelled as point-to-point links. The connection between DERS, Attacker and the Switch are modelled as a CSMA link. CSMA link is the closest possible connection type to the Ethernet connection in the NS-3. This Ethernet type connection is required to carryout the Man-In-The-Middle attacks using ARP spoofing. Router R1 acts as the gateway for the DERs to reach the Aggregator 1. We used the UDP sockets as the transport layer protocol for the communication. As one can see in Figure 3, the nodes DER1, DER2, DER3, DER4, Aggregator1, DSO and the RTU Grid1 exists in both NS-3 and the RTDS simulator. This is because the data is actually generated and consumed at the RTDS Simulator. NS-3 is just there to make the packets under go simulated network conditions before they reenter the RTDS simulator. We have made some modification to the original NS-3 version 3.29 so that it is able to carry out attacks such as ARP spoofing, TCP SYN flood, Capturing the packet at IP level and making modifications. These modifications are explained in the Section 7 and the source code can be made available for those who are interested. In the

Networking simulation conducted, we have used the emulated networking interface of NS-3.

3.3. Interfacing RTDS with NS-3

The NS-3 simulator need to run on a Linux machine, which has two or more network interface cards. The network connectivity between the RTDS and this Linux machine is achieved through the GTNET card of the RTDS simulator. The connection setup is shown in Figure 4.

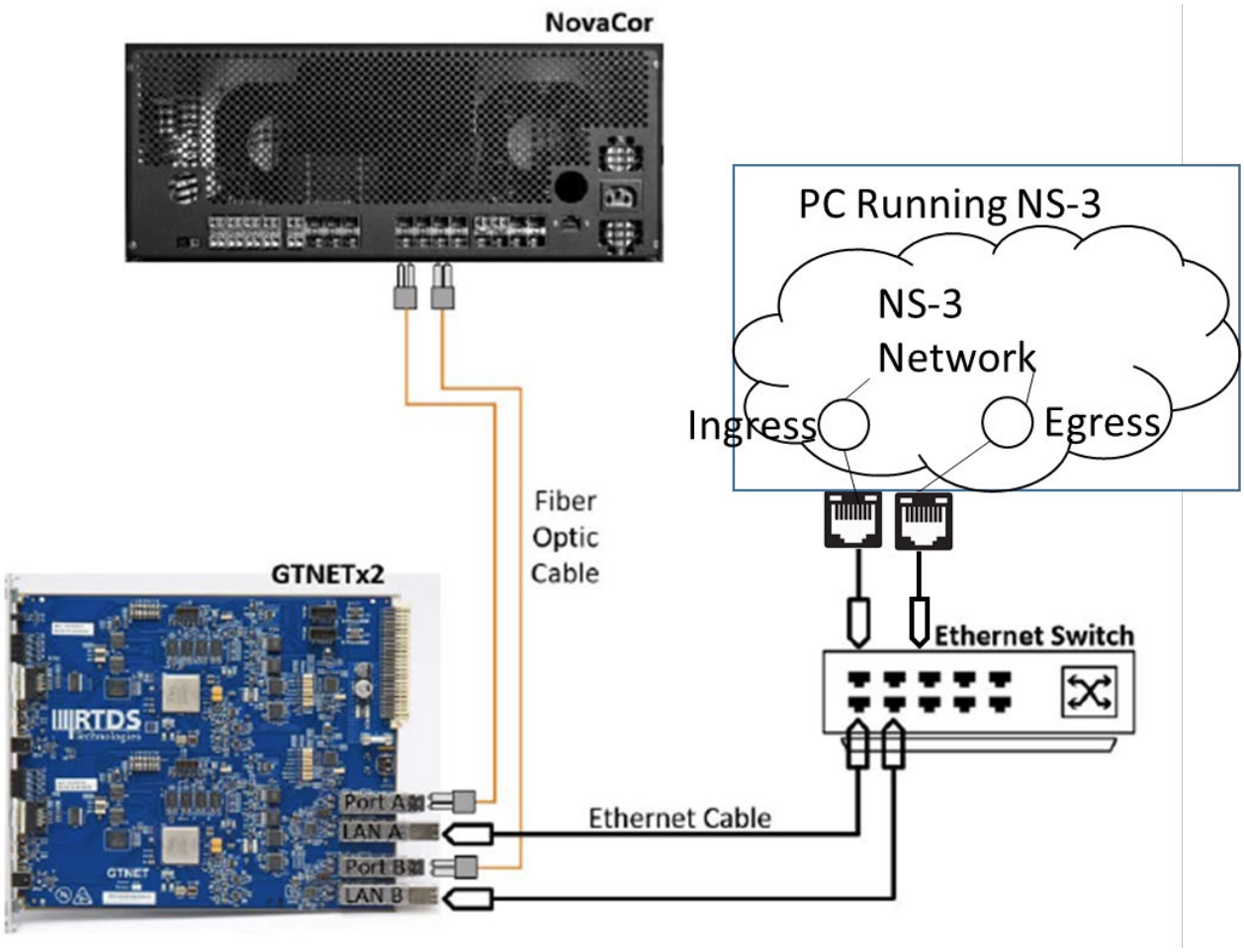


Figure 4 Interfacing the RTDS with NS-3

Here, as shown on Figure 4, the simulated nodes Ingress and egress are directly connected to each of the Ethernet ports on the machines. These nodes acts as the entry and exit points

to and from the simulated network. For an example if we want the data to traverse from DER1 to the Aggregator, the flow of data happens in the following order:

1. The Data is calculated at the RTDS simulator
2. GTNET interface corresponding to DER1 sends data to the Ingress interface
3. Ingress node forwards the data to the simulated DER1 Node
4. The data travers the network and gets to the simulated Aggregator node
5. Simulated aggregator node sends the data to the Egress node
6. Egress node forwards the data to the GTNET interface corresponding to the Aggregator node at RTDS simulator.

The other communication between the nodes happen in a similar fashion.

4 PEAK SHAVING APPLICATION

In this application all the calculations are performed using a script running in the RSCAD. The data flows of this application are as follows:

1. The control system reads measurement data from Remote Terminal Units (RTU) in the field (e.g. in substations) and delivers the data to the distribution management system (DMS).
2. A state estimator in the DMS calculates power flow estimates for all grid assets. If any of the assets are loaded above the limit, the DMS calculates the inverted difference as a reference signal.
3. The DMS sends a reference signal to one or several aggregators. In the case where several Aggregators are jointly providing the service, the signal will be split and be sent to all contracted aggregators corresponding to each aggregator's proportional share in the installed capacity or service commitment.
4. The aggregators requests flexibility information from all DER units in its portfolio.
5. The DER units respond with a flexibility prognosis.
6. The aggregator performs an internal optimization of its portfolio, in order to be able to deliver the service in the cheapest and most optimal way.
7. The aggregator sends set-points to all connected units and requests flexibility updates.
8. The DER units respond with an updated flexibility prognosis.
9. Smart meters at the DER owner provide measurements to the DSO.

In the simulation case, we simulate a system with a single Aggregator. The following calculations are carried out using the runtime script in the RTDS simulator:

1. Calculation of the reference signal in step 2

   a. Here we collect the information of the current loading by reading a meter in runtime in the script

   b. Then we calculate the level of overloading using a set threshold in the script

2. Calculation of the flexibility for each DER at step 5

   a. Here, we use a set percentage for the increment in power output and load reduction

3. Calculation of the set-points at the aggregator in step 6

4.1. Running Denial of Service (DoS) and Man-In-The-Middle attacks (MITM)

We developed some capability into NS-3 to run attacks such as DoS and Man-In-The-Middle. The DoS attack can result in the server going unresponsive to the legitimate traffic. Furthermore, the receive queues of the nodes are limited in size. Therefore the legitimate traffic might get completely dropped or get delayed. In the case of MITM, the attacker changes the packets with a malicious intent in mind. In our simulated scenario, the DoS attacks on the Aggregator resulted in the distribution transformer being overloaded for an extended period of time. In the case of MITM attack, we assumed a scenario where the attacker work in favor of DER1. Therefore, the attacker made changes to the flexibility information of the other DERs such that they do not have any flexibility. This resulted in DER1 unfairly getting to sell all of its flexibility and earning the maximum.

4.1.1. DoS attacks in NS-3

The construction of DoS attacks in NS-3 differs based on the transport layer protocol used. For the UDP transport layer, the DoS attack is just one or many nodes sending a lot of bogus UDP traffic at a given port of the server, which is listening on that port. This makes the server overwhelmed with the traffic. In the case of TCP transport layer, DoS attacks are created by sending many SYN packets with bogus source IPs. The SYN packet is the first packet sent to establish a TCP session. Then the server allocates resources and send the SYN ACK packet. However, since the SYN packet has a bogus source IP there is no one to accept the SYN ACK. Then the server keeps this session open for a given time and then close it. When there are a larger number of packets like this, the server can go out of resources for the legitimate traffic. This type of DoS attack is called a SYN flood attack.

4.1.2. MITM attacks in NS-3

In the testing carried out, we used ARP spoofing in order to carry out MITM attacks. In order for the ARP spoofing to work, both the attacker and the victim should be in the same sub-net. ARP protocol maps the IP addresses to the Medium access control (MAC) address. When a node needs to send a packet to an IP address which is in its own sub-net it requests the MAC address of the node which bears the given IP address. This request is called the ARP request. This request is broadcasted in the sub-net. Then the node bearing that IP replies with the MAC address. This is called the ARP reply. This mapping between the MAC address and the IP address is stored in the ARP table of this node. These entries have a time of expiry.
In the case of ARP spoofing, the attacker listens to these ARP requests and replies with its own MAC address. Then the victim sends the data packets to the attacker without knowing that this is a malicious user. Then the attacker can either extract information before sending it to the intended user or it can modify or drop that packet. Another way of achieving the same result is sending unsolicited ARP replies or ARP requests. Figure 5 shows the first scenario.

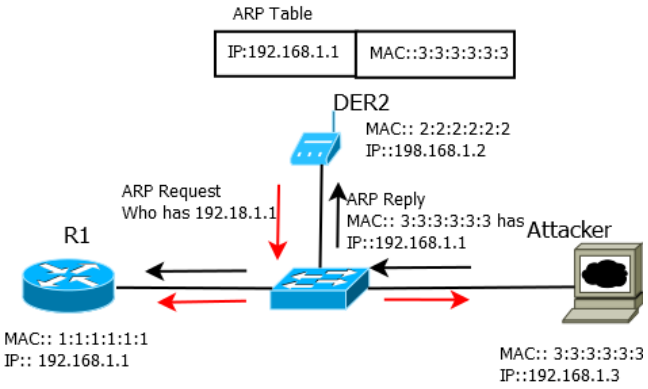


Figure 5 ARP spoofing

4.2. Running the case CyberSecSimPeakShaveUDP in RTDS

In this section, we describe the steps involved in running this case in the RTDS simulator. The sample case can be made available to those interested. This case uses the GTNET SKT-Multi firmware. Therefore, the SKT 1.23 firmware or above must be installed.

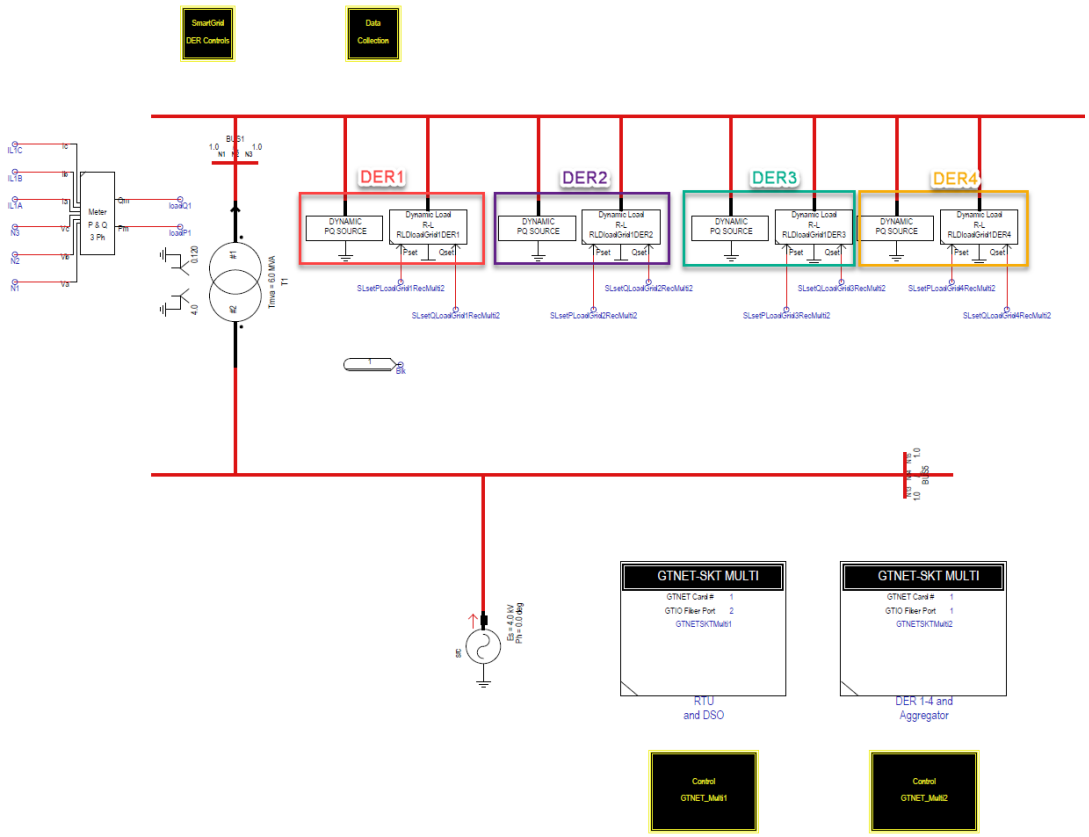


Figure 6 Power system model for SKT UDP simulation

4.2.1. Changes needed in the Draft

We first discuss the changes needed in the GTNET card used by the RTU and the DSO. Here we used 2 channels. One used by the RTU and the other used by the DSO. For this communication, we did not use the NS-3 network as an intermediate hop. This communication was directly between the two GTNET channels where RTU is the client and the DSO is the server. This is not a limitation. We can use the NS-3 for this communication as well. We did this to lessen the complexity of the NS-3 simulation. Each of the channels of the GTNET card needs an assigned IP. These can be set at the Config file editor in the RSCAD main window. Let us assume that the IP address of the RTU is 172.24.9.249 and that of DSO is 172.24.9.250. Then, the remote address of the RTU should be set to the IP address of the DSO which is 172.24.9.250. The DSO is sending correction signal to the Aggregator. Therefore, the remote IP of DSO is set to the IP address of the Aggregator responsible for DSO. Let us call it Aggregator-DSO interface. Aggregator-DSO interface has the IP address 172.24.9.248 in the example case. The 3rd channel in the RTU and DSO GTNET interface is used to get the information on the time the distribution transformer stays overloaded. The information

is captured in NS-3. The remote IP of this channel is set to the IP of the Ingress port which is 172.24.2.139 in the example case.

Now, let us look at the changes needed in the GTNET card used by the DERs and the aggregator. First the IP addresses should be set for the DERs. Let us assume we use the IPs 172.24.9.240-172.24.9.243 for the DERs. In order to be able to process this control scenario in the script we used five IP addresses for the Aggregator. Four of these are used for the communication with each DER and the other for the communication with the DSO. Had we did all this processing using a C++ code inside the aggregator node in NS-3 we could eliminate the need for all of these IPs since the aggregator node processing is no longer carried out using a script. The other way of doing this is aggregating all the DER flexibility. Let us assume the aggregator use 172.24.9.244 for the communication with DER1 (let us call it Aggregator-DER1), 172.24.9.245 for Aggregator-DER2, 172.24.9.246 for Aggregator-DER3, 172.24.9.247 for Aggregator-DER4 and 172.24.9.248 for Aggregator-DSO. Then let us assume that the IP address of the ingress interface of the machine running NS-3 is 172.24.2.139 and that of the egress interface is 172.24.2.102.

Since we need to send the traffic between the DERs and the Aggregator through the simulated network, we set the remote IPs of the DERs and Aggregator as follows:

- DER1 remote IP 172.24.2.139
- DER2 remote IP 172.24.2.139
- DER3 remote IP 172.24.2.139
- DER4 remote IP 172.24.2.139
- Aggregator-DER1 remote IP 172.24.2.139
- Aggregator-DER2 remote IP 172.24.2.139
- Aggregator-DER3 remote IP 172.24.2.139
- Aggregator-DER4 remote IP 172.24.2.139

The other parameters set in the GTNET multi are the variables received and transmitted. These are specified in the “From GTNET-SKT-x ” and “To GTNET-SKT-x” settings respectively. The ingress node is capable of finding out which DER or Aggregator it should forward the packet. This is made possible by including the DER index and the Aggregator index in the data sent over the GTNET card.

4.2.2. Running the case in Runtime

This case is run using the script provided. In the script we set the P and Q load and P and Q generation as shown in the below table.

Node name	P Load	P Generation	Q Load	Q Generation
DER 1	4.5 MW	3.0 MW	4.0 MVar	3.0 MVar
DER 2	4.5 MW	3.0 MW	4.0 MVar	3.0 MVar
DER 3	4.5 MW	3.0 MW	4.0 MVar	3.0 MVar
DER 4	4.5 MW	3.0 MW	4.0 MVar	3.0 MVar

The P load at the Distribution transformer is maintained at 4.8MW and 3.6 MVar, using load shedding and increased generation at the DERs. The DSO send the correction signal to Aggregator to reduce the P load by 1.2 MW and Q load by 0.4 MVar. Each DER is assumed to be able to reduce the load by 80% and increase the generation by 20%. DERs send this information to the Aggregator. Then the aggregator calculates the set points and send them to the DERs. As mentioned before these calculations are carried out in the script. To repeat the process to get the average values we reset the values of the loads and generation of the DERs to the initial value after the script go through the loop twice in the RSCAD script. Furthermore, we pick DERs in random order in each iteration to use their flexibility. Therefore, every DER is able to sell their flexibility with equal opportunity. The time the transformer remain overloaded is indicated on the meter, timeOverloadPlot2, in runtime.

4.3. Running the UDP Cyber-security simulation in NS-3

We have created an example NS-3 scenario at "`~/ns3_cybersec/ns-allinone-3.29/ns-3.29/examples/RTDS-DoS-Simulatio/rtds-dos-simulation_UDP_BiDir.cc`". This scenario can be run by navigating to the "`~/ns3_cybersec /ns-3-allinone/ns-3.29`" folder and running the command below. However, remember that the IP addresses for DERs and Aggregators should be the ones that you set on the GTNET cards. The IP addresses for the Int1IP and the Int2IP are the IP addresses of the network interface cards of the PC running NS-3.

- `NS_LOG="Icmpv4L4Protocol":"Ipv4Protocol" ./waf --run "rtds-dos-simulation_UDP_BiDir --stopTime=500 --DoSEnabled=false --ArpSpoofEnabled=false --IPDER1=172.24.9.10 --IPDER2=172.24.9.11 --`

```
IPDER3=172.24.9.12 --IPDER4=172.24.9.13 --IPAggreDER1=172.24.9.14 --
IPAggreDER2=172.24.9.15 --IPAggreDER3=172.24.9.16 --
IPAggreDER4=172.24.9.17 --Int1IP=172.24.2.101 --Int2IP=172.24.2.102 –
Int1MAC=08:00:27:48:57:c3  --Int2MAC=08:00:27:20:6d:04 --
Gateway=172.24.0.1 --Subnet=2 -- deviceName1=enp7s0 --
deviceName2=enp8s0”
```

Now, let us break this command into parts and see what each subpart does.

- `NS_LOG="Icmpv4L4Protocol":"Ipv4Protocol"` : This shows all the information on the Icmpv4 protocol and IPv4 protocol when NS-3 simulation enters the code related to those protocols
- `./waf --run "rtds-dos-simulation_UDP_BiDir`: This part starts the simulation stated in the C++ code rtds-dos-simulation_UDP_BiDir.cc
- `--stopTime=500` : These are the options. This option set the stop time of the simulation to be 500 seconds
- `--DoSEnabled=false` : When true this starts a DoS attack at 100 seconds and ending at 200 seconds.
- `--ArpSpoofEnabled=false`: When true this enables ARP spoofing and we set the flexibility of all the DERs except DER1 to 0.0
- `--IPDERx`=172.24.9.10 : Here x can be 1,2,3 or 4. This make the egress node send the data directed to DERx to this IP address.
- `--IPAggreDERx`=172.24.9.14: Here x can be 1,2,3 or 4. This make the egress node send to data directed to Aggregator-DERx to this IP address
- `--Int1IP=172.24.2.101`: This sets the IP address of the ingress node to this IP
- `--Int2IP=172.24.2.102`: This sets the IP address of the egress node to this IP
- `--Int1MAC=08:00:27:48:57:c3`: This sets the MAC address of the ingress node to the given MAC
- `--Int2MAC=08:00:27:20:6d:04`: This sets the MAC address of the egress node to the given MAC
- `--Gateway=172.24.0.1` : This sets the gateway of the second interface
- `--Subnet=2`: This sets the subnet mask for the two NS-3 emu interfaces

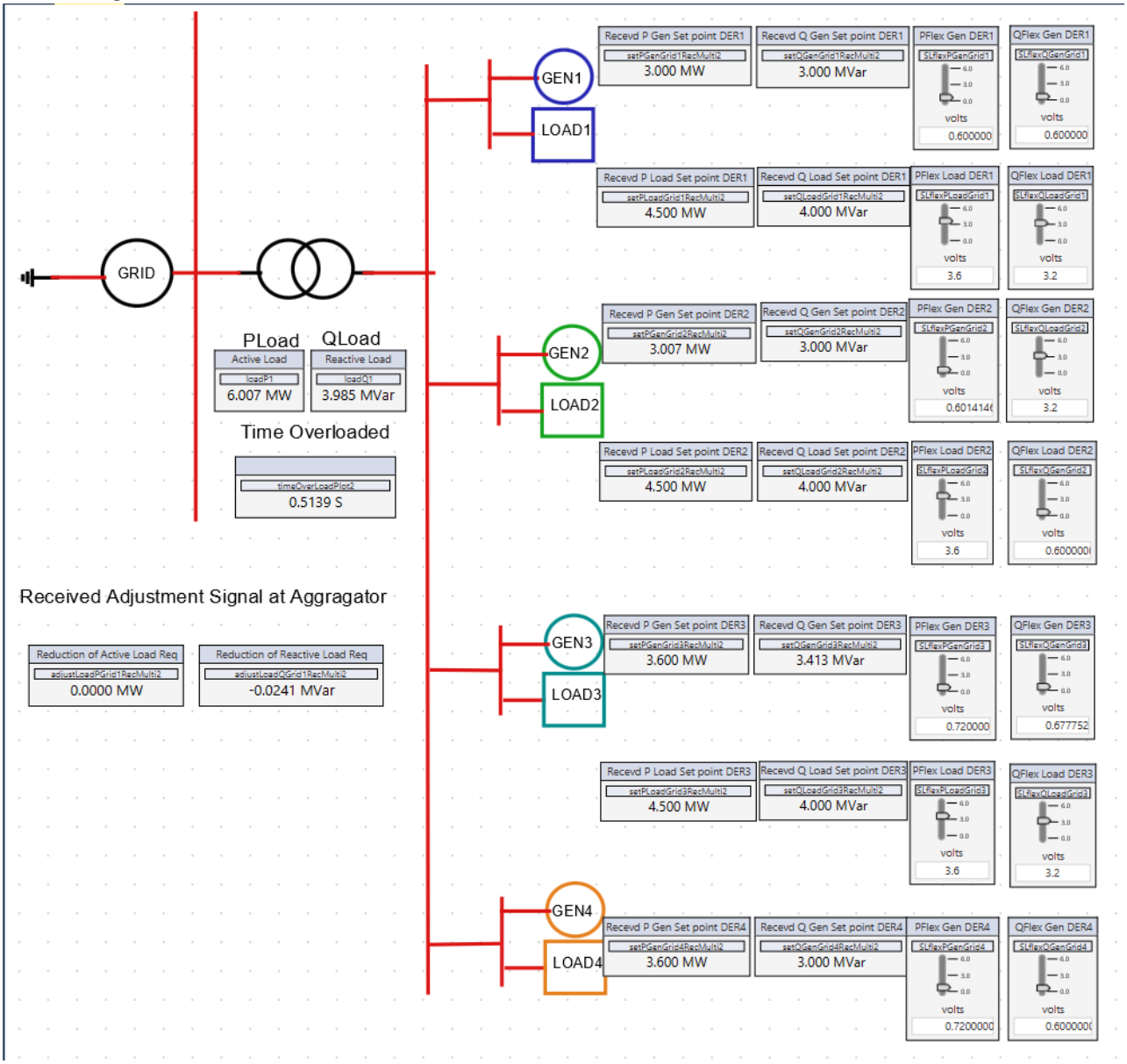
Running this command starts the ns-3 simulation. However, before this command is run both the network interfaces in the Linux machine running NS-3 should be in the promiscuous mode. This can be run by running the following Linux command.

- `sudo ifconfig <interface_name> promisc`

The interfaces can be listed using the command: `ifconfig -a`

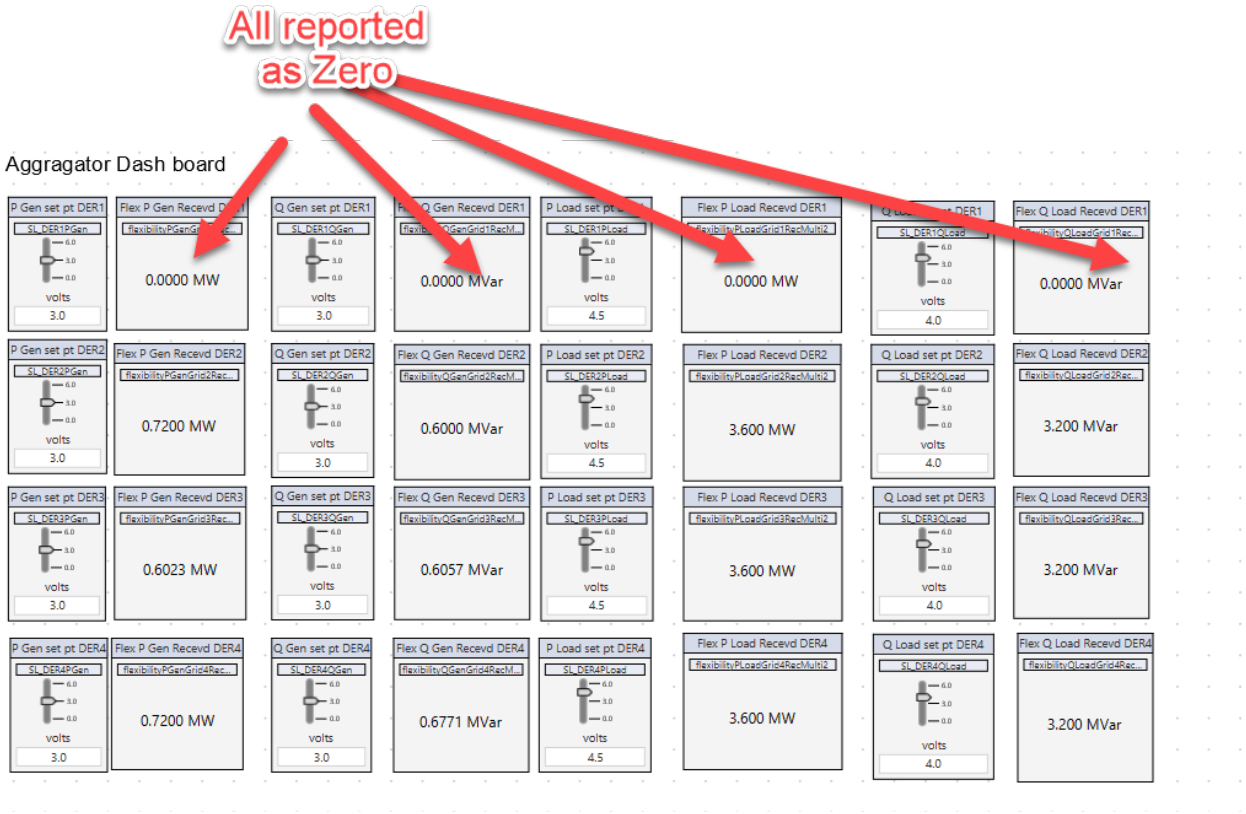
4.3.1. Running the Co-simulation

If you run the RSCAD simulation and the NS-3 simulation with out attacks you would see something similar to the image below:

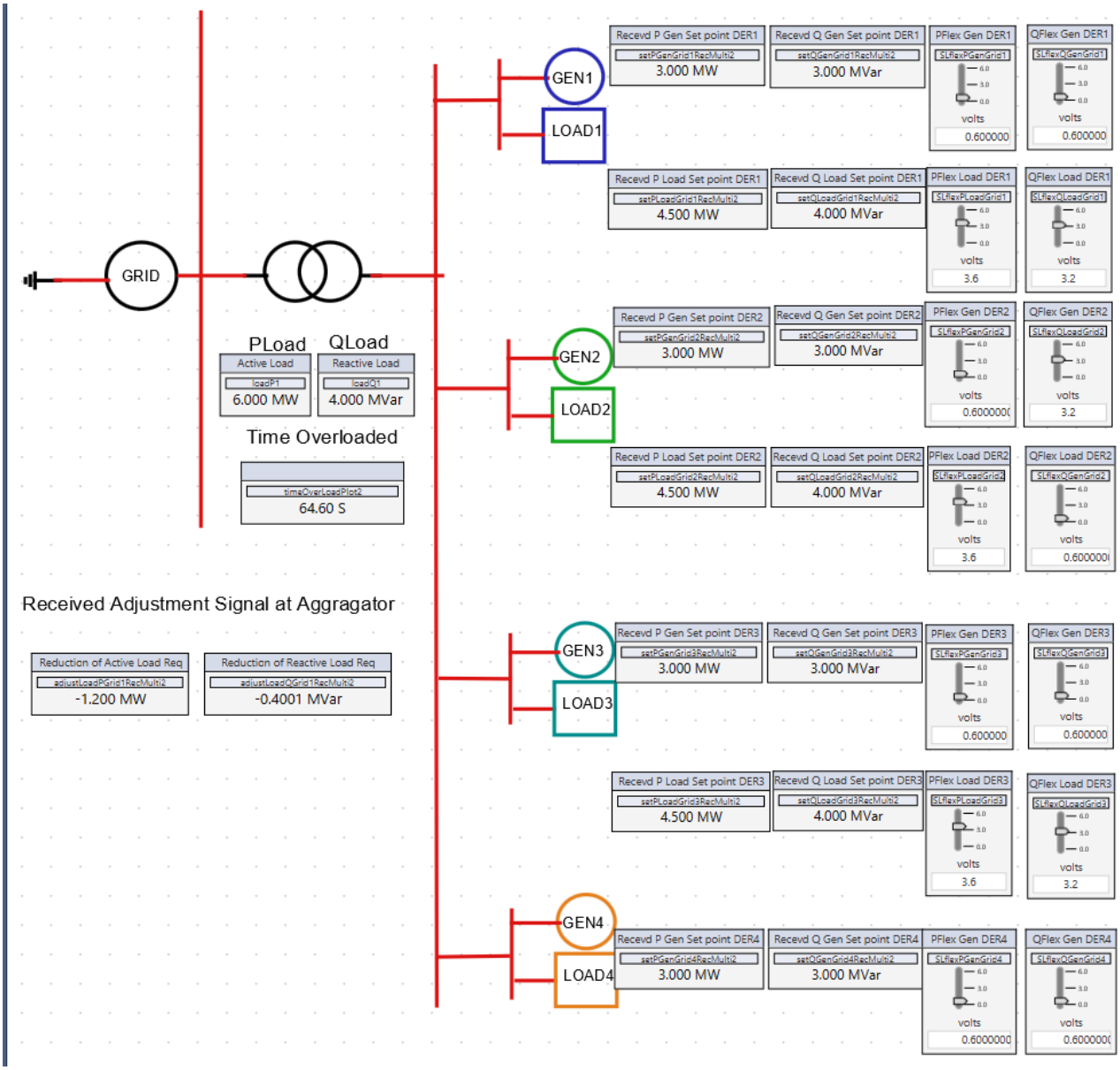


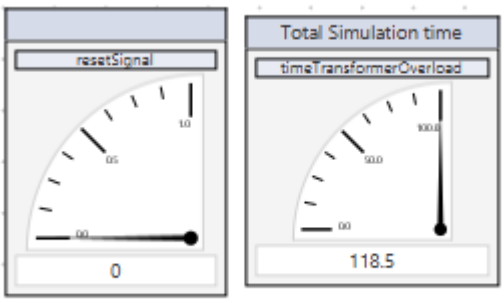
If you observe it for a sufficiently long time you would see that all the three DER have equal opportunity in

selling their flexibility to the Aggregator. In the next simulation we are going to make the flexibility of the DER1 zero using a Man-In-The-Middle attack. Then you would see that only the other DERs are able to sell their flexibility.



Now let us run a Denial of service attack at the Aggregator. Here, we are sending a lot of UDP traffic at the Aggregator so that it is unable to process the actual traffic. This will make the distribution transformer being overloaded for a longer time. We will start this attack 40 sec into the NS-3 simulation and end it at 50 sec into the simulation. However, the system do not recover.


GRID
PLoad
Active Load
loadP1
6.000 MW
QLoad
Reactive Load
loadQ1
4.000 MVar
Time Overloaded
timeOverLoadPlot2
64.60 S
Received Adjustment Signal at Aggragator
Reduction of Active Load Req
adjustLoadPGrid1RecMulti2
-1.200 MW
Reduction of Reactive Load Req
adjustLoadQGrid1RecMulti2
-0.4001 MVar
GEN1
LOAD1
Recevd P Gen Set point DER1
setPGenGrid1RecMulti2
3.000 MW
Recevd Q Gen Set point DER1
setQGenGrid1RecMulti2
3.000 MVar
PFlex Gen DER1
SLflexPGenGrid1
volts
0.600000
QFlex Gen DER1
SLflexQGenGrid1
volts
0.600000
Recevd P Load Set point DER1
setPLoadGrid1RecMulti2
4.500 MW
Recevd Q Load Set point DER1
setQLoadGrid1RecMulti2
4.000 MVar
PFlex Load DER1
SLflexPLoadGrid1
volts
3.6
QFlex Load DER1
SLflexQLoadGrid1
volts
3.2
GEN2
LOAD2
Recevd P Gen Set point DER2
setPGenGrid2RecMulti2
3.000 MW
Recevd Q Gen Set point DER2
setQGenGrid2RecMulti2
3.000 MVar
PFlex Gen DER2
SLflexPGenGrid2
volts
0.600000
QFlex Gen DER2
SLflexQLoadGrid2
volts
3.2
Recevd P Load Set point DER2
setPLoadGrid2RecMulti2
4.500 MW
Recevd Q Load Set point DER2
setQLoadGrid2RecMulti2
4.000 MVar
PFlex Load DER2
SLflexPLoadGrid2
volts
3.6
QFlex Load DER2
SLflexQGenGrid2
volts
0.600000
GEN3
LOAD3
Recevd P Gen Set point DER3
setPGenGrid3RecMulti2
3.000 MW
Recevd Q Gen Set point DER3
setQGenGrid3RecMulti2
3.000 MVar
PFlex Gen DER3
SLflexPGenGrid3
volts
0.600000
QFlex Gen DER3
SLflexQGenGrid3
volts
0.600000
Recevd P Load Set point DER3
setPLoadGrid3RecMulti2
4.500 MW
Recevd Q Load Set point DER3
setQLoadGrid3RecMulti2
4.000 MVar
PFlex Load DER3
SLflexPLoadGrid3
volts
3.6
QFlex Load DER3
SLflexQLoadGrid3
volts
3.2
GEN4
LOAD4
Recevd P Gen Set point DER4
setPGenGrid4RecMulti2
3.000 MW
Recevd Q Gen Set point DER4
setQGenGrid4RecMulti2
3.000 MVar
PFlex Gen DER4
SLflexPGenGrid4
volts
0.600000
QFlex Gen DER4
SLflexQGenGrid4
volts
0.600000


resetSignal
0
Total Simulation time
timeTransformerOverload
118.5

4.4. Running the case CyberSecSimPeakShaveTCP in RTDS

This case is similar to the case where we used the UDP transport layer protocol. In this case, we used TCP only for the communication between DERs and the Aggregator, since we are planning to disrupt this communication inside the simulated NS-3 network using cyber attacks. However, when running TCP it was difficult to use a single IP address for both reception and transmission. Therefore, we reduced the case to have only two DERs. Each one of these DERs has one IP to listen to incoming connections and the other for initiating a connection. Similarly, we have two IPs for the Aggregator-DER1 interface and Aggregator-DER2 interface.

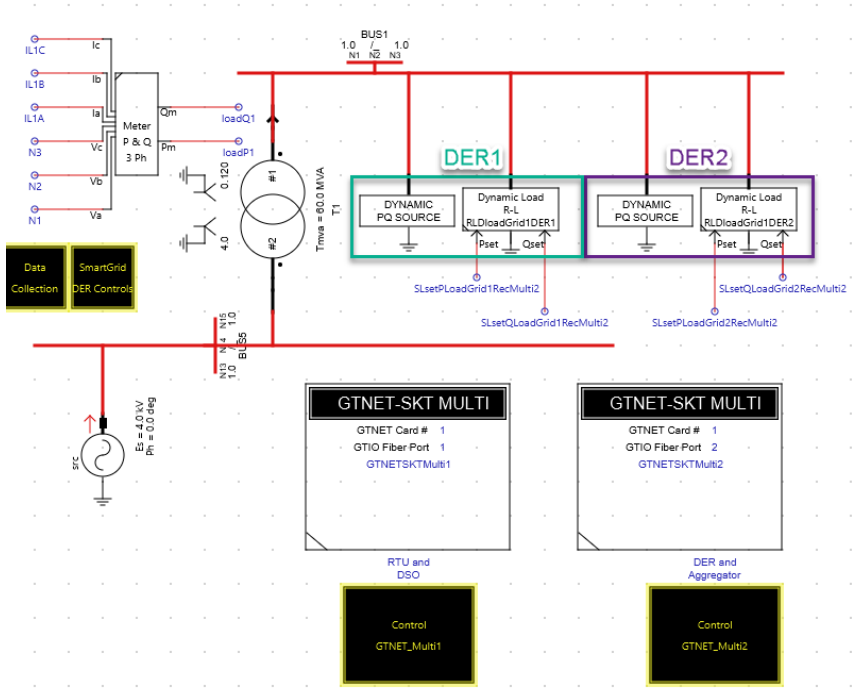


Figure 7 Draft case for the RSCAD simulation

This case is run using the script provided. In the script we set the P and Q load and P and Q generation as shown in the below table.

Node name	P Load	P Generation	Q Load	Q Generation
DER 1	4.5 MW	3.0 MW	4.0 MVar	3.0 MVar
DER 2	4.5 MW	3.0 MW	4.0 MVar	3.0 MVar

The P load at the Distribution transformer is maintained at 2.4 MW and 1.8 MVar using load shedding and increased generation at the DERs. The DSO send the correction signal to Aggregator to reduce the P load by 0.6 MW and Q load by 0.2 MVar. Each DER is assumed to be able to reduce the load by 80% and increase the generation by 20%. The other details are the same as the case having the UDP transport layer protocol.

4.5. Running the TCP Cyber-security simulation in NS-3

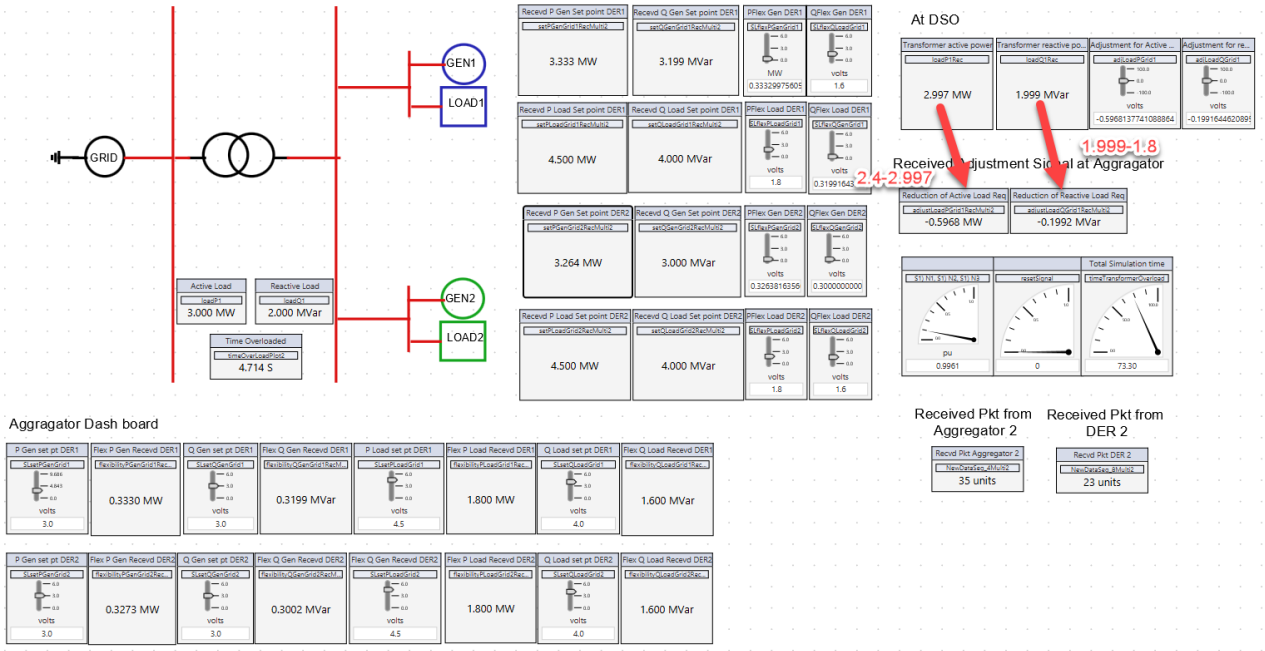
We have created an example NS-3 scenario at "`~/ns3_cybersec/ns-allinone-3.29/ns-3.29/examples/RTDS-DoS-Simulation/rtds-dos-simulation_TCP_BiDir.cc`". This scenario can be run by navigating to the `~/ns3_cybersec /ns-3-allinone/ns-3.29` folder and running the command below. However, remember that the IP addresses for DERs and Aggregators should be the ones that you set on the GTNET cards. The IP addresses for the Int1IP and the Int2IP are the IP addresses of the network interface cards of the PC running NS-3.

- `NS_LOG="Icmpv4L4Protocol":"Ipv4Protocol" ./waf --run "rtds-dos-simulation_TCP_BiDir --stopTime=500 --DoSEnabled=false --ArpSpoofEnabled =false --IPDER1C=172.24.9.10 --IPDER1S=172.24.9.11 --IPDER2C=172.24.9.12 --IPDER2S=172.24.9.13 --IPAggreDER1S=172.24.9.14 --IPAggreDER1C=172.24.9.15 --IPAggreDER2C=172.24.9.16 --IPAggreDER2S=172.24.9.17 --Int1IP=172.24.2.101 --Int2IP=172.24.2.102 --InterSynTime=1e-6 --maxParallelSessions=100 --Int1MAC=08:00:27:48:57:c3 --Int2MAC=08:00:27:20:6d:04 --Gateway=172.24.0.1 --Subnet=2 -- deviceName1=enp7s0 --deviceName2=enp8s0"`

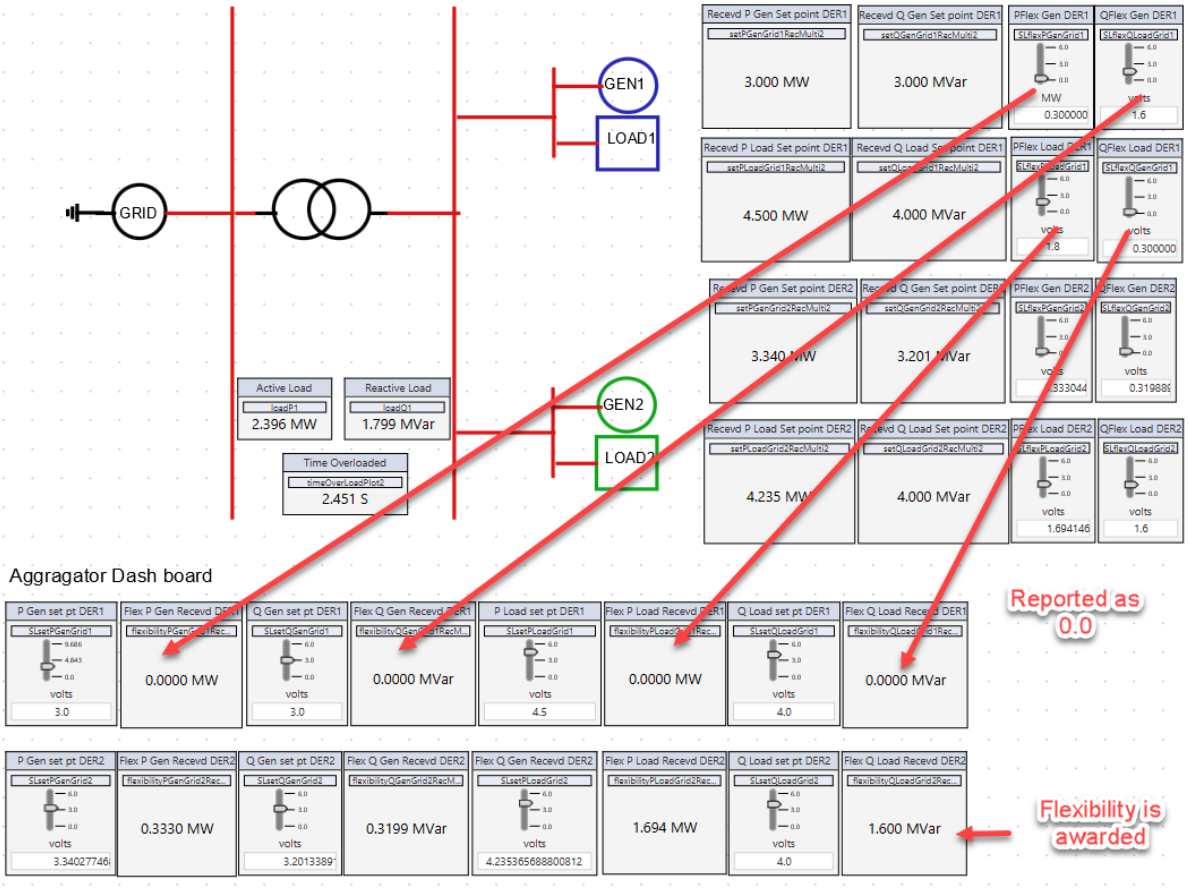
Here, the IPDERxS, where x is 1 or 2, is the server interface of DERs 1 and 2. The IPDERxC is the client interface of the DERs. IPAggreDERxS is the interface IP of the aggregator which listens to packets from the DERs. The IPAggreDER1C is the interface which sends packets to the DERs. The option InterSynTime defines the time between two consecutive SYN packets when running the SYN flood DoS attack. The option maxParallelSessions is the number of TCP sessions that can be served by the node concurrently. All the other options are same as the scenario with UDP transport layer protocol.

4.5.1. Running the Co-simulation

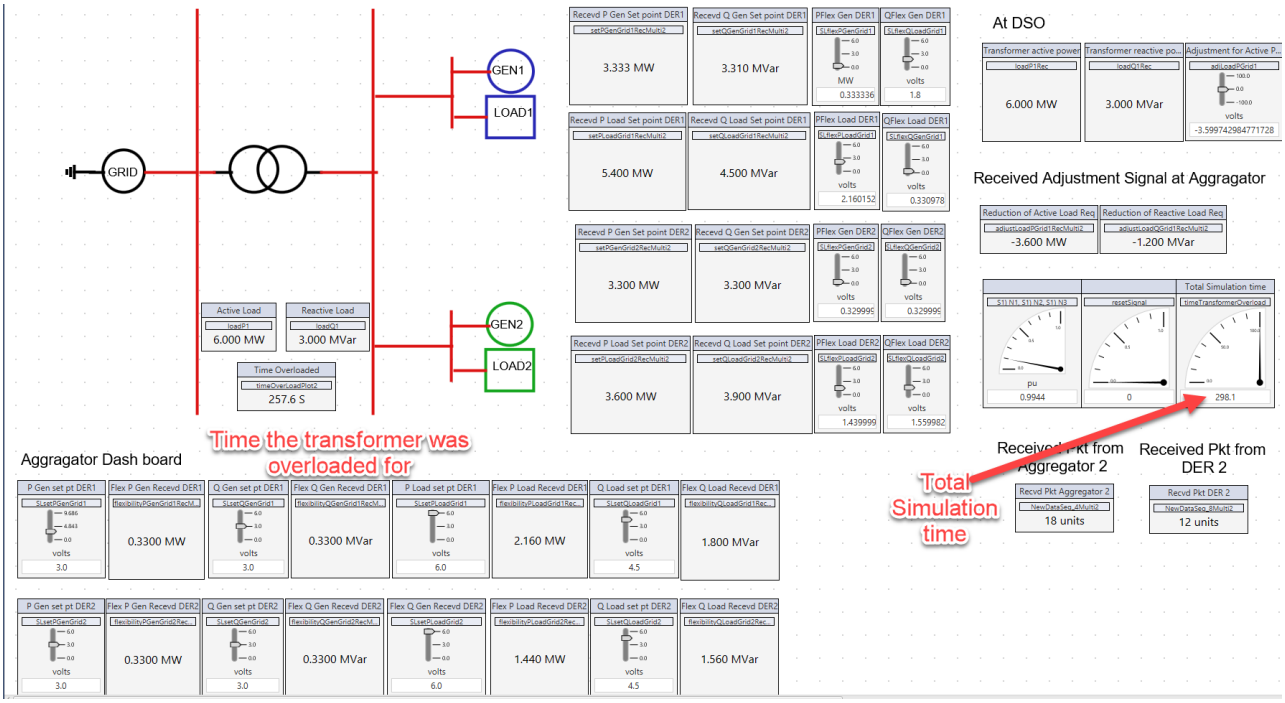
If you observe the RSCAD runtime without any attack in NS-3, you would see something similar to the figure below. It will not stay static. However, from time to time it will show something similar the below image. It will change since the RSCAD script loops through the entire process multiple times.



If we run a Man-In-The-Middle attack and make DER1 have no flexibility, you would see something similar to the image below:



If we run a Syn-flood attack on the Aggregator, the entire process will be halted at 50sec into the simulation. NS-3 does not seem to recover from it although we stop the DoS attack at 60 Sec.



If you compare the total time the simulation ran and the time the transformer was at over loaded state you will see that the time is approximately 50 sec. This is given that you started the NS-3 and RSCAD script approximately at the same time.
You will also see that the NS-3 stops writing to the console at around 50 sec.

```
At Ingress Type:7
Index of Node:1
Sent to:10.1.6.18
In Aggregator
Type:7Index of Node:2Sent to:10.1.7.4
Aggregator Sent to:10.1.7.4port:7001
At aggregator
DER Sent to
10.1.7.4-->10.103.41.241
Egress at=48.5566s, rx bytes=28
Egress Sent to:10.103.41.124
```

4.5.2. Modifying the iptable rules

The iptables in Linux is the firewall of the Linux machine. It is possible for it to block incoming tcp ports. Furthermore, it will send RST packets or icmp port unreachable packets when a SYN packet arrives to ports ns-3 nodes are listening to (ex: tcp port 7001). Therefore, the following commands should be executed (ignore the statements starting with #).

```
#Dropping RST
sudo iptables -I OUTPUT -p tcp --tcp-flags RST RST -j DROP


# Accept the packets sent to tcp port 7001
 sudo iptables -A INPUT -p tcp -m tcp --dport 7001 -m conntrack --ctstate
NEW,UNTRACKED -j ACCEPT
```

4.5.3. Make the rules permanent

Install the iptables-save package to save the iptable rules to make them permanent using the following command.

```
sudo apt-get install iptables-persistent
```

After making changes to the iptables enter the following command to make the rules permanent.

```
sudo iptables-save
```

5 USING NS-3 TO MODIFY DNP3 PACKETS

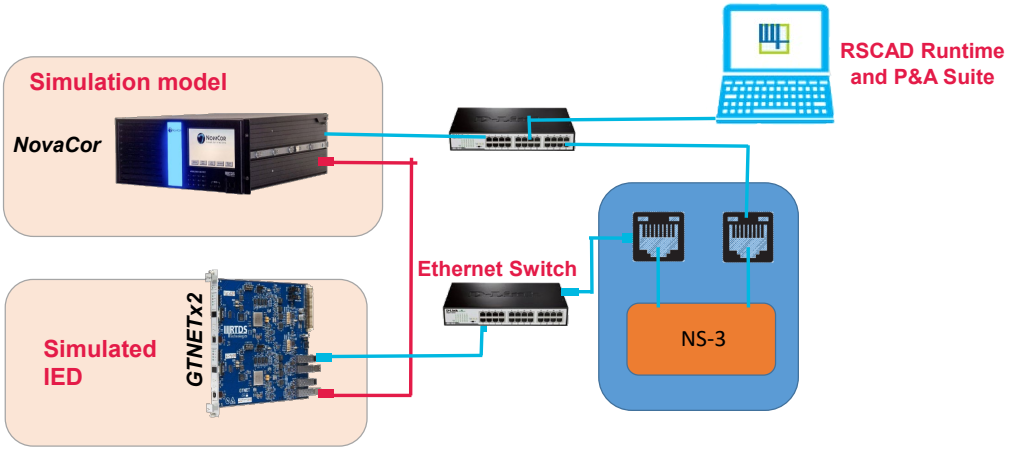


Figure 8 Network setup for NS-3 simulation

We use the above network topology when connecting the GTNET card to the P&A suite or any other controller. Here, the NS-3 simulation creates a simulated network between the GTNET card and the controller or the monitor. We can run Man-In-The-Middle attacks and Denial of service attacks within the NS-3 network.

5.1. Running the simulation in NS-3

This simulation is run using the TAP interface of NS-3. First we need to create two TAP interfaces and two bridge interfaces in the Linux machine running NS-3. Let the tap interfaces be tap00 and tap01 and the bridge interfaces be br0 and br1. Then, we bridge one of the physical Ethernet interfaces (say enp7s0) with tap00 using br0. Then we bridge the other physical interface (say enp8s0) with tap01 using br1. The set of commands to be entered are given below:

```
sudo ip link add name br0 type bridge
sudo ip link add name br1 type bridge

sudo ip tuntap add mode tap tap00
sudo ip tuntap add mode tap tap01

sudo ip link set dev tap00 address 00:00:00:00:01:20
sudo ip address add 0.0.0.0 dev tap00
sudo ip link set dev tap00 promisc on
```

```
sudo ip link set dev tap00 up

sudo ip link set dev tap01 address 00:00:00:00:01:21
sudo ip address add 0.0.0.0 dev tap01

sudo ip link set dev enp7s0 down
sudo ip address add 0.0.0.0 dev enp7s0
sudo ip link set dev enp7s0 up

sudo ip link set dev enp8s0 down
sudo ip address add 0.0.0.0 dev enp8s0
sudo ip link set dev enp8s0 up

sudo ip link set dev enp11s0 down
sudo ip address add 0.0.0.0 dev enp11s0
sudo ip link set dev enp11s0 up


sudo ip link set dev tap01 promisc on
sudo ip link set dev tap01 up

sudo ip link set dev tap00 master br0
sudo ip link set dev enp7s0 master br0


sudo ip link set dev tap01 master br1
sudo ip link set dev enp8s0 master br1

sudo ip link set dev br0 up
sudo ip link set dev  br1 up
sudo ip link set dev  enp7s0 promisc on
sudo ip link set dev  enp8s0 promisc on
```

A bash script that contains these instructions are included in `~/ns3_cybersec/ns-allinone-3.29/ns-3.29/setupInterface`. To run this script navigate to the folder `~/ns3_cybersec/ns-allinone-3.29/ns-3.29/` and enter `./setupInterface` at the terminal.

The simulation file we use for this simulation can be found in `~/ns3_cybersec/ns-allinone-3.29/ns-3.29/examples/RTDS-DoS-Simulation/rtds-Tap-ICS-Mod-One_Net.cc`

5.2. Decoding the simulation file

The simulation script is a C-file which can be found at `~/ns3_cybersec/ns-allinone-3.29/ns-3.29/examples/RTDS-DoS-Simulation/rtds-Tap-ICS-Mod-One_Net.cc`. Here we explain only the important elements in the file. For more information the users are referred to the official wiki page of NS-3, https://www.nsnam.org/wiki/Main_Page.

First we have a set of includes which are in the NS3 API. We created the application Attack-app to help the users in creating a Man-In-The-Middle type of attack. The header file for it is `#include "ns3/attack-app.h"`. We also made some modifications to the ARP protocol in NS3 to facilitate the ARP poisoning type of attack. The details about these modifications are listed in Chapter 11.4. The include statements are shown below:

```
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <iostream>
#include <fstream>
#include "ns3/ipv4-address-generator.h"
#include <string>
#include <cassert>
#include <iomanip>
#include "ns3/command-line.h"
#include "ns3/config.h"
#include "ns3/uinteger.h"
#include "ns3/boolean.h"
#include "ns3/double.h"
#include "ns3/string.h"
#include "ns3/log.h"
#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/udp-client-server-helper.h"
#include "ns3/log.h"
#include "ns3/ipv4-address.h"
#include "ns3/nstime.h"
#include "ns3/inet-socket-address.h"
#include "ns3/inet6-socket-address.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include "ns3/netanim-module.h"
```

```
#include "ns3/udp-socket-factory.h"
#include "ns3/packet-sink-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/fd-net-device-module.h"
#include "ns3/point-to-point-module.h"
#include <sys/socket.h>
#include <arpa/inet.h>
#include "ns3/flow-monitor-helper.h"
#include "ns3/csma-helper.h"
```

Then inside the main function, first we set some global parameters so that we can connect the virtual NS3 network to the real world. The first parameter is the `SimulatorImplementationType`, which specifies whether it is a Real time simulator implementation or not. The second parameter is `ChecksumEnabled`, which will add the IP checksum to the packets sent.

```
GlobalValue::Bind ("SimulatorImplementationType", StringValue
("ns3::RealtimeSimulatorImpl"));
GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
```

Next, we create a set of Nodes. A node can be a mobile device, a PC, a switch, a router or a firewall. However, note that they are just empty hulls that does not do anything. The user will need to either code what it is supposed to do by mean of an NS3 application or use an existing application in NS3.

```
// Create ns3 nodes for simulating the communications network
  Ptr<Node> n0 = CreateObject<Node> ();
  Ptr<Node> n1 = CreateObject<Node> ();
  Ptr<Node> n2 = CreateObject<Node> ();
  Ptr<Node> n3 = CreateObject<Node> ();
  Ptr<Node> n4 = CreateObject<Node> ();
```

In this example, we use a very simple network setting. All the virtual nodes and the external nodes are in the same CSMA network. CSMA is a protocol similar to Ethernet. However, there are differences between the two. NS3 does not have an Ethernet implementation build in. Since all the nodes are going to be on the same network, we next create a container that contains all the nodes.

```
NodeContainer n0n1n2n3 = NodeContainer (n0, n1, n2, n3, n4);
```

Afterwards, we create a CSMA network with a data rate of 100Mbps and an end-to-end delay of 6560 Nano Seconds. Then we install this CSMA network interfaces in the above-created nodes. All these network devices (you can assume they are Ethernet ports), are saved in the NetDeviceContainer dn0n1n2n3.

```
CsmaHelper csmaNetwork;
csmaNetwork.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csmaNetwork.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
NetDeviceContainer dn0n1n2n3 = csmaNetwork.Install (n0n1n2n3);
```

The next step is to install the network stack on the nodes created. This consists of the ARP protocol, IP protocol and the socket protocol installed in the nodes. Without these protocols these nodes cannot send or receive packets.

```
// installing IP stacks into the nodes
  InternetStackHelper stack;
  stack.Install (n0);
  stack.Install (n1);
  stack.Install (n2);
  stack.Install (n3);
  stack.Install (n4);
```

Next we assign IP addresses to these nodes using the Ipv4AddressHelper class. When assigning the IP addesses we make sure that the IPs of the outside world devices are in the same network.

```
Ipv4AddressHelper ipv4;
ipv4.SetBase ("172.24.0.0", "255.255.0.0", "0.0.9.241"); //this is the outer net-
work
Ipv4InterfaceContainer ipn0n1n2n3 = ipv4.Assign (dn0n1n2n3);
```

If we need specific routes set they should be set next. However, since this example does not

need any special routing we are not setting any here. Please refer to the official NS3 Wiki if you need information on it.

Next, we create the interfaces which will be connected to the outside world. We create interfaces on the nodes n0 and n4 that can be connected to the outside world. Then we can assume node n0 and node n4 are our outside world nodes. We are connecting the tap00 to node n0 and tap 01 to node n4. We created these tap devices in the Linux system using the script in Section 5.

```
// Use the TapBridgeHelper to connect to the pre-configured tap devices for
// the left side.  We go with "UseBridge" mode since the CSMA devices support
// promiscuous mode and can therefore make it appear that the bridge is
// extended into ns-3.  The install method essentially bridges the specified
// tap to the specified CSMA device.
// Connect the left side tap to the left side CSMA device in ghost node n0

  TapBridgeHelper tapBridge;
  tapBridge.SetAttribute ("Mode", StringValue ("UseBridge"));
  tapBridge.SetAttribute ("DeviceName", StringValue ("tap00"));
  tapBridge.Install (n0, dn0n1n2n3.Get (0));
  tapBridge.SetAttribute ("DeviceName", StringValue ("tap01"));
  tapBridge.Install (n4, dn0n1n2n3.Get (4));
```

Now that the infrastructure is complete, we can install the applications on the nodes. Here, we only install the attack-app that we created. This application will help us run the man-in-the-middle attack. First, we pick n3 to be our attacker node who will do ARP spoofing and modify the packets. Then we get the IP address and the interface index of the CSMA interface of the attacker node. Afterwards, we get a pointer to the Ipv4Interface of the CSMA interface of node n3.

```
uint32_t attackerId = 3;
std::pair<Ptr<Ipv4>, uint32_t> returnValue = ipn0n1n2n3.Get (attackerId);
Ptr<Ipv4> ipv4Val = returnValue.first;
uint32_t index = returnValue.second;
Ptr<Ipv4Interface> iface = ipv4Val->GetObject<Ipv4L3Protocol> ()->GetInterface
(index);
```

The next step is configuring the attacker-app. Here, first we create the AttackApp object and

save the pointer to it in attacker object. Then we create vectors of the IP addresses to be spoofed, the victims IP address and the victims MAC address as spoofedIPs, victimIPs and victimMACs respectively. When users with the IP addresses in the victimIPs vector try to access the IP addresses in the spoofedIPs vector their ARP table mapping of the MAC address for the spoofedIPs will be replaced by the MAC address of the Attacker. This is achieved by sending unsolicited ARP replies to the node with MAC addresses in victimMACs. An image showing how this is done is shown in Figure 5.

Next we setup the attacker-app using the Setup command. The arguments of this function are the node the application is installed, the network interface to use, the IPv4 interface, the spoofedIPs, victimIPs and the victimMACs. Then we add the attacker application to the node n3. Finally we start the application with a scheduled start and end times.

```
Ptr<AttackApp> attacker = CreateObject<AttackApp> ();
std::vector<Ipv4Address> spoofedIPs{Ipv4Address ("172.24.9.251")};
std::vector<Ipv4Address> victimIPs{Ipv4Address ("172.24.9.55")};
std::vector<Address> victimMACs{ns3::Mac48Address ("00:0A:35:00:10:09")};

attacker->Setup (n0n1n2n3.Get (attackerId), dn0n1n2n3.Get (attackerId), iface,
spoofedIPs, victimIPs, victimMACs);
n0n1n2n3.Get (attackerId)->AddApplication (attacker);
attacker->SetStartTime (Seconds (1.0));
attacker->SetStopTime (Seconds (3600.0));
```

Next we do the same procedure for the traffic sent in the reverse direction using the code below. This is the end of the applications setup.

```
std::pair<Ptr<Ipv4>, uint32_t> returnValue2 = ipn0n1n2n3.Get (attackerId);
Ptr<Ipv4> ipv4Val2 = returnValue2.first;
uint32_t index2 = returnValue2.second;

Ptr<Ipv4Interface> iface2 = ipv4Val2->GetObject<Ipv4L3Protocol> ()->GetInterface
(index2);

Ptr<AttackApp> attacker2 = CreateObject<AttackApp> ();
std::vector<Ipv4Address> spoofedIPs1{Ipv4Address ("172.24.9.55")};
std::vector<Ipv4Address> victimIPs1{Ipv4Address ("172.24.9.251")};
std::vector<Address> victimMACs1{ns3::Mac48Address ("10:65:30:05:d8:ff")};

attacker2->Setup (n0n1n2n3.Get (attackerId), dn0n1n2n3.Get (attackerId), iface,
spoofedIPs1,victimIPs1, victimMACs1);
n0n1n2n3.Get (attackerId)->AddApplication (attacker2);
attacker2->SetStartTime (Seconds (1.0));
attacker2->SetStopTime (Seconds (3600.0));
```

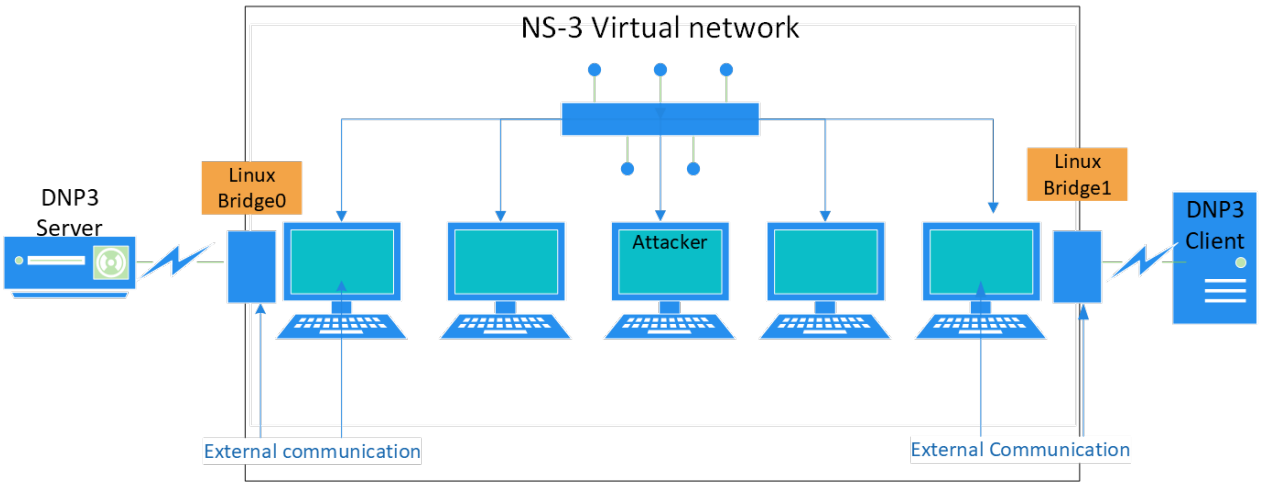
Then we enable packet capturing in all the interfaces in the CSMA network using the below command.

```
csmaNetwork.EnablePcapAll ("pmuconnectiontestNet", false);
```

Finally we schedule the simulation to run for 1 hour and deallocate all memory at the end of it using the below commands.

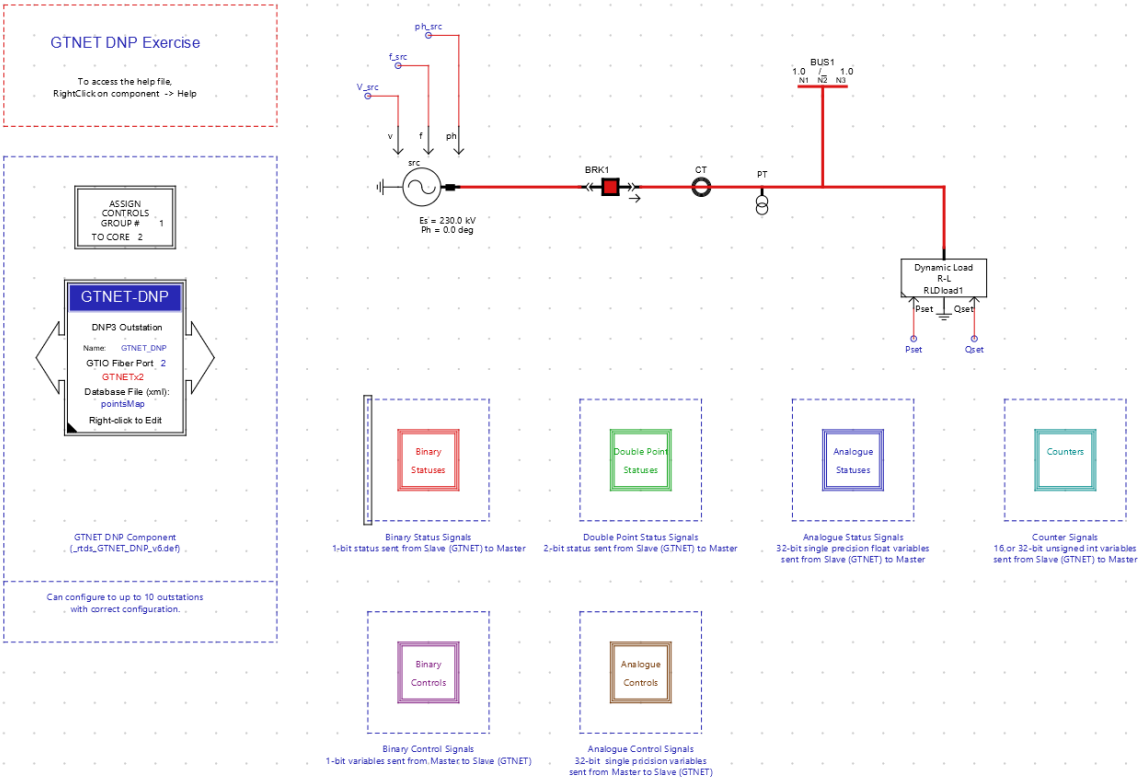
```
// Run the simulation for 1 hour to give the user time to play around
//
Simulator::Stop (Seconds (3600.));
Simulator::Run ();
Simulator::Destroy ();
```

5.3. Virtual NS-3 network created

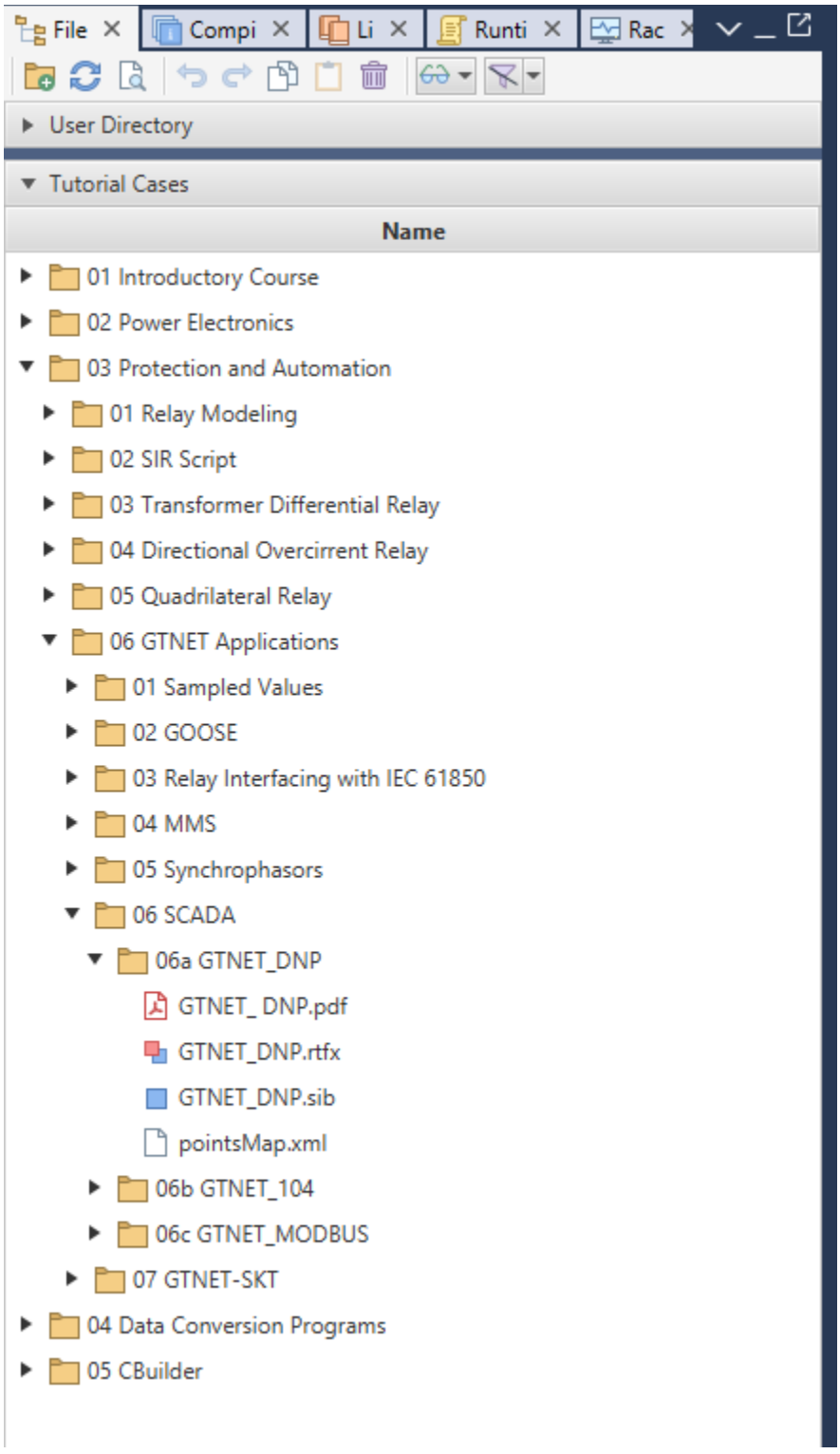


Using the above code we create the above network. The external communication is achieved by bridging the physical Ethernet device with a virtual tap device created in Linux. The NS-3 nodes are able to interface with those Tap devices. Hence, the traffic will be routed in the Virtual CSMA network before exiting through the bridge on the other side. The attacker node sends Spoofed ARP replies to the DNP3 server and the DNP3 client so that the traffic will be routed through it.

5.4. Example scenario: GTNET_DNP



The above case can be found in the Tutorials tab in RSCAD FX 2.0 at the following location:



Here, we have a simple network with a power source and a dynamic load with a breaker in the middle. The user can control the breaker status, and the set active power and reactive power in the dynamic load using DNP3. Here we use the P&A suite to control these

parameters.

5.1. Running the NS-3 DNP3 simulation

To run the NS-3 Tap simulation to modify DNP3 data, first navigate to the ns-3-allinone/ns-3.29. Then execute the following command:

```
NS_LOG="Icmpv4L4Protocol":"Ipv4Protocol"  ./waf  --run  "rtds-Tap-ICS-Mod-One_Net"
```

This will start the NS-3 simulation.

5.2. Setting and Monitoring DNP3 values

In the runtime of the RSCAD simulation you are able to see the current that travels in the bus and the set active and reactive power of the dynamic load. Then in the P&A suite you are able to connect to the DNP3 server hosted on the GTNET cards. Furthermore, you can modify and monitor the parameters using DNP3 communication. If there is a man-in-the-middle attack on DNP3 communication, there will be a disparity between what you see in the runtime and in the P&A suite.

5.3. Stating the new values to be assigned for DNP3

The modifications to the DNP3 data are included in a file "`ns3.conf`". This file should be located in `/etc/ns3` in the Linux machine. The format of the text are given below

```
protocol dnp3
<function_code>  <Group> <Variation> <Index> <value>
```

Here the file parser uses the protocol dnp3 to identify that the following data are for dnp3 packet modification. The next lines contain information on what to modify.

Function_code - The Function code of the dnp3 message of interest

Group - Is the DNP3 group number of the parameter

Variation - Is the DNP3 variation of the group of the parameter

Index - Is the index of the variable belonging to the stated group and variance

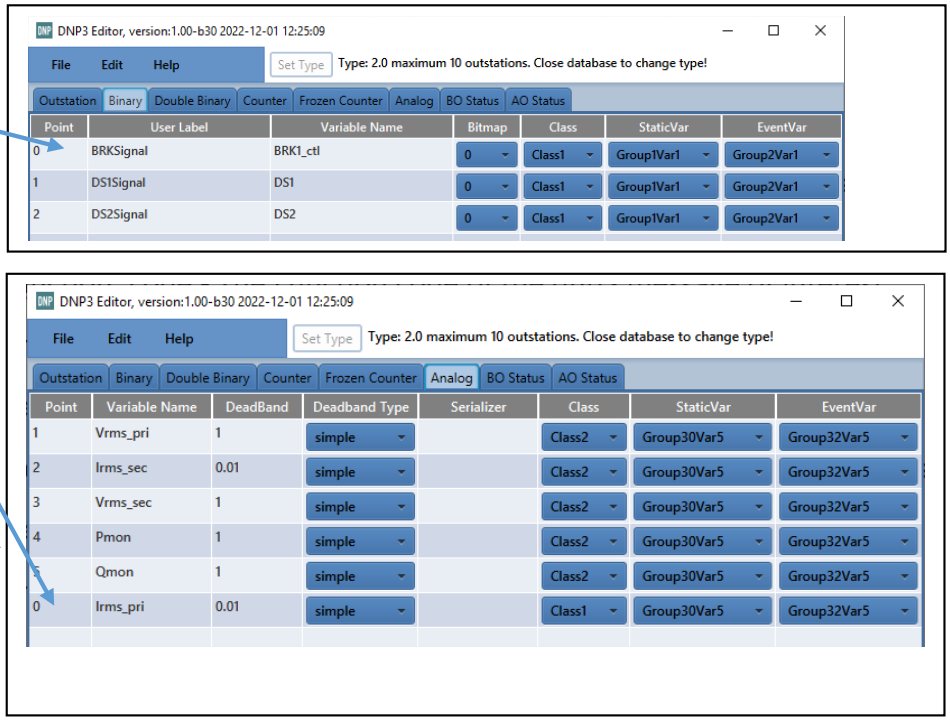
Value - Is the new parameter value to be assigned

An example is shown below:

```
protocol dnp3
129 1 1 0 0
129 30 5 0 100
129 32 5 0 100
3 12 1 0 3
4 12 1 0 3
129 12 1 0 4
3 41 3 0 200
4 41 3 0 200
129 41 3 0 150
129 1 1 0 0
129 40 3 0 23
```

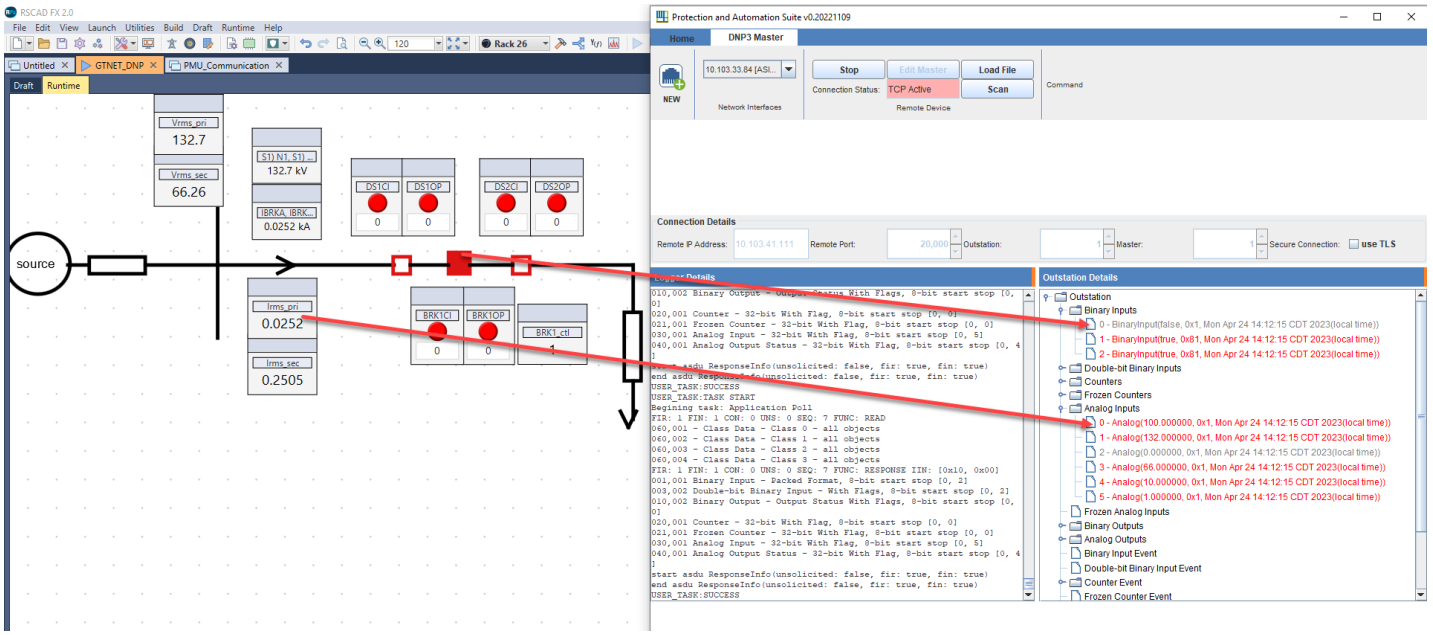
Manipulate LATCH_OFF

Manipulate first Analog control block



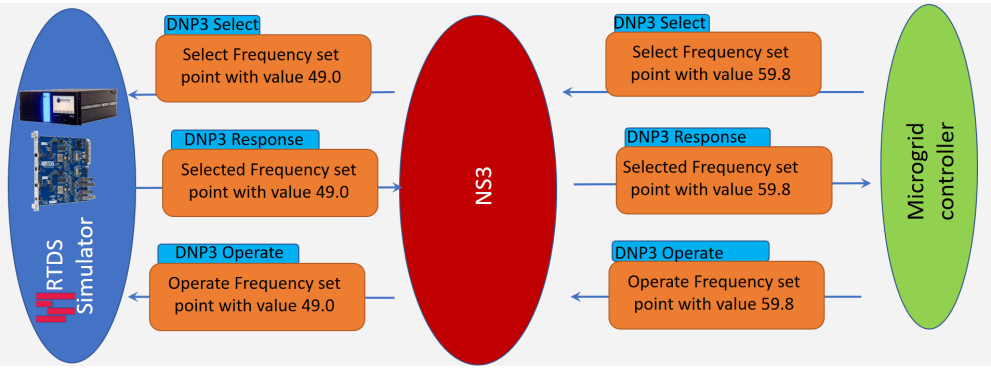
As shown above the first 3 lines manipulate the reported Binary status of the breaker and the Analog input corresponding to the Irms_pri. Here, the function code used is 129 which corresponds to the Response function of the DNP3 server. Using the first line we modify the first binary input status to OPEN. Using the second line we modify the first analog input to 100.

After running the simulation the following can be observed in the runtime.



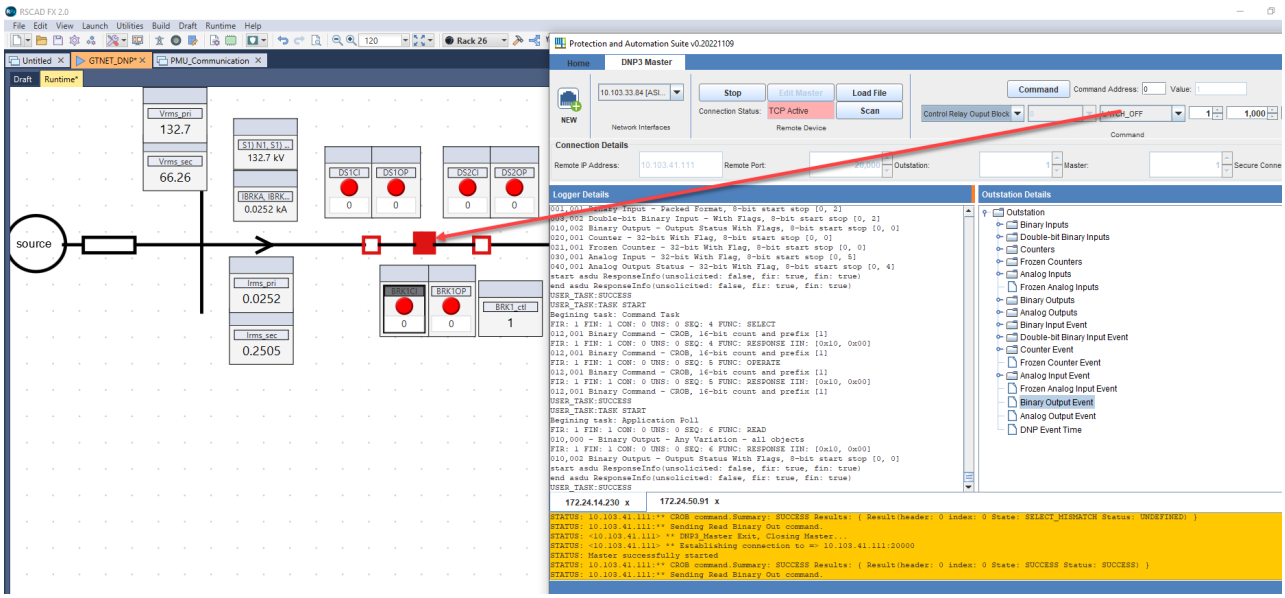
Here, you can see that the breaker status is shown as open although it is closed. Furthermore, the value of Irms_pri is reported as 100.0 when the actual value is 0.02524

The commands in DNP3 usually happens in 3 stages as shown below:

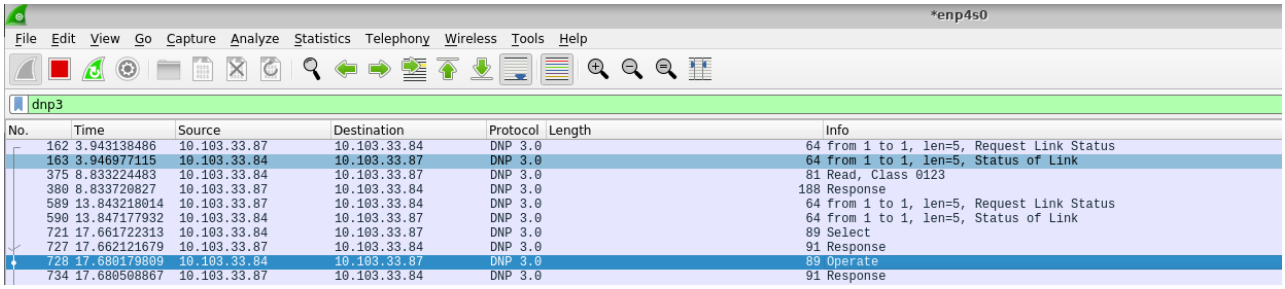


The Select function has a value of 3, the Operate function has value of 4 and the response function has a value of 129. In the lines 3, 4 and 5 of the above example after the line, protocol dnp3, we are modifying a command, which instructs to latch off a breaker. Using the values set for the SELECT and OPERATE function we will not let the breaker latch off. The value passed by the DNP3 master for the LATCH_ON operation is three. Using the first line, 3 12 1 0 3, we modify the LATCH_OFF (value of 4) value to LATCH_ON (value of 3) for the variable with index 0 of Group number 12 and variance 1. This modification is done for the SELECT message. Then we do the same for the OPERATE message in the second line. Then in line 5 we modify the RESPONSE message to carry the value of LATCH_OFF. We do this because the DNP3 master does not send the OPERATE command if the SELECT value and the

RESPONSE values are different. Make sure you use the port 20000 as the Listening port at DNP3 server (The GTNET-DNP3).



You will see that although you send the LATCH_OFF command it will not be reflected in the Runtime. If you see a packet capture at the client interface you will see that there is nothing wrong there.

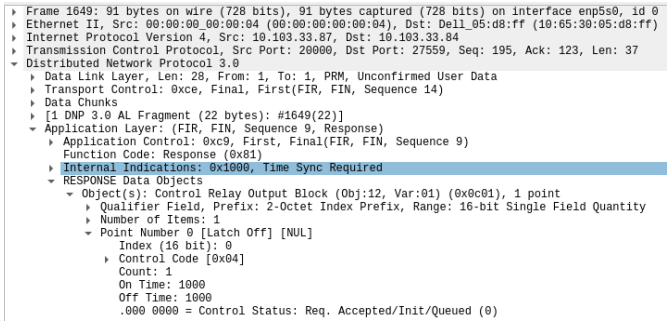


Client Side server side	Server side
Payload for SELECT	Payload for SELECT

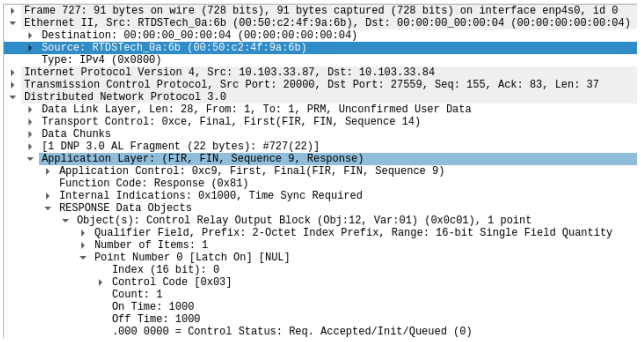
```
Frame 1643: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface enp5s0, id 0
Ethernet II, Src: Dell_05:d8:ff (10:65:30:05:d8:ff), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
Internet Protocol Version 4, Src: 10.103.33.84, Dst: 10.103.33.87
Transmission Control Protocol, Src Port: 27559, Dst Port: 20000, Seq: 88, Ack: 195, Len: 35
Distributed Network Protocol 3.0
    Data Link Layer, Len: 26, From: 1, To: 1, DIR, PRM, Unconfirmed User Data
    Transport Control: 0xc9, Final, First(FIR, FIN, Sequence 9)
    Data Chunks
    [1 DNP 3.0 AL Fragment (20 bytes): #1643(20)]
    Application Layer: (FIR, FIN, Sequence 9, Select)
        Application Control: 0xc9, First, Final(FIR, FIN, Sequence 9)
        Function Code: Select (0x03)
    SELECT Request Data Objects
        Object(s): Control Relay Output Block (Obj:12, Var:01) (0x0c01), 1 point
            Qualifier Field, Prefix: 2-Octet Index Prefix, Range: 16-bit Single Field Quantity
            Number of Items: 1
            Point Number 0 [Latch Off] [NUL]
                Index (16 bit): 0
                Control Code [0x04]
                Count: 1
                On Time: 1000
                Off Time: 1000
                .000 0000 = Control Status: Req. Accepted/Init/Queued (0)
```

```
Frame 721: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface enp4s0, id 0
Ethernet II, Src: 00:00:00_00:00:04 (00:00:00:00:00:04), Dst: RTDSTech_0a:6b (00:50:c2:4f:9a:6b)
    Destination: RTDSTech_0a:6b (00:50:c2:4f:9a:6b)
    Source: 00:00:00_00:00:04 (00:00:00:00:00:04)
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.103.33.84, Dst: 10.103.33.87
Transmission Control Protocol, Src Port: 27559, Dst Port: 20000, Seq: 48, Ack: 155, Len: 35
Distributed Network Protocol 3.0
    Data Link Layer, Len: 26, From: 1, To: 1, DIR, PRM, Unconfirmed User Data
    Transport Control: 0xc9, Final, First(FIR, FIN, Sequence 9)
    Data Chunks
    [1 DNP 3.0 AL Fragment (20 bytes): #721(20)]
    Application Layer: (FIR, FIN, Sequence 9, Select)
        Application Control: 0xc9, First, Final(FIR, FIN, Sequence 9)
        Function Code: Select (0x03)
    SELECT Request Data Objects
        Object(s): Control Relay Output Block (Obj:12, Var:01) (0x0c01), 1 point
            Qualifier Field, Prefix: 2-Octet Index Prefix, Range: 16-bit Single Field Quantity
            Number of Items: 1
            Point Number 0 [Latch On] [NUL]
                Index (16 bit): 0
                Control Code [0x03]
                Count: 1
                On Time: 1000
                Off Time: 1000
                .000 0000 = Control Status: Req. Accepted/Init/Queued (0)
```

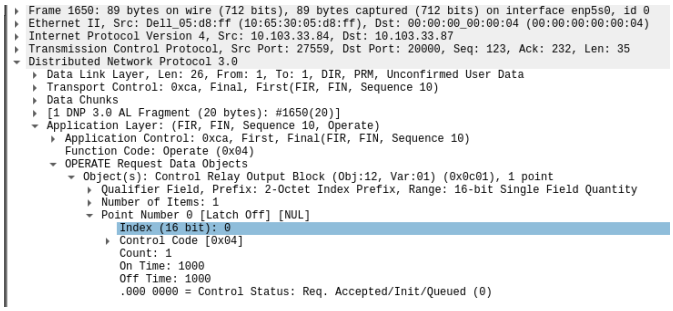
Payload for Response



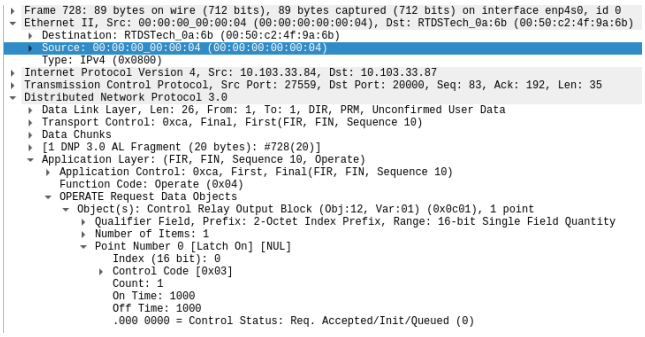
Payload for Response



Payload for Operate



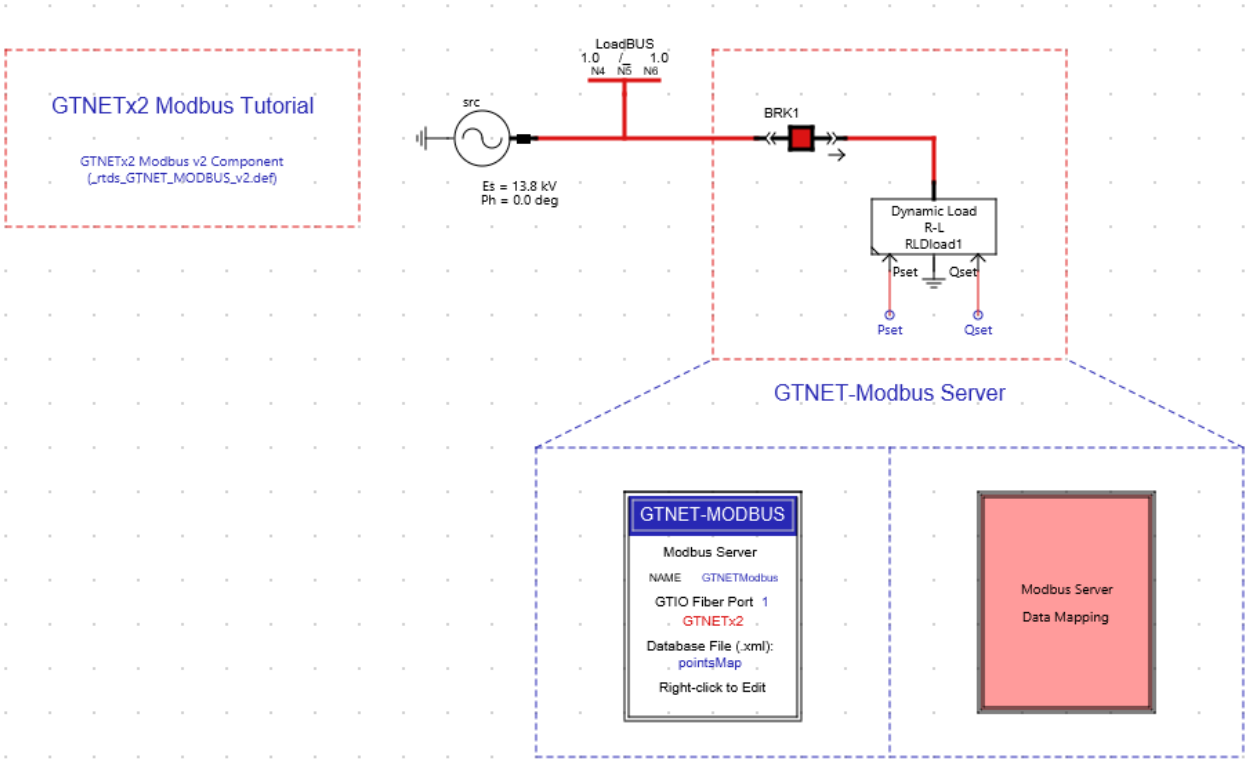
Payload for Operate



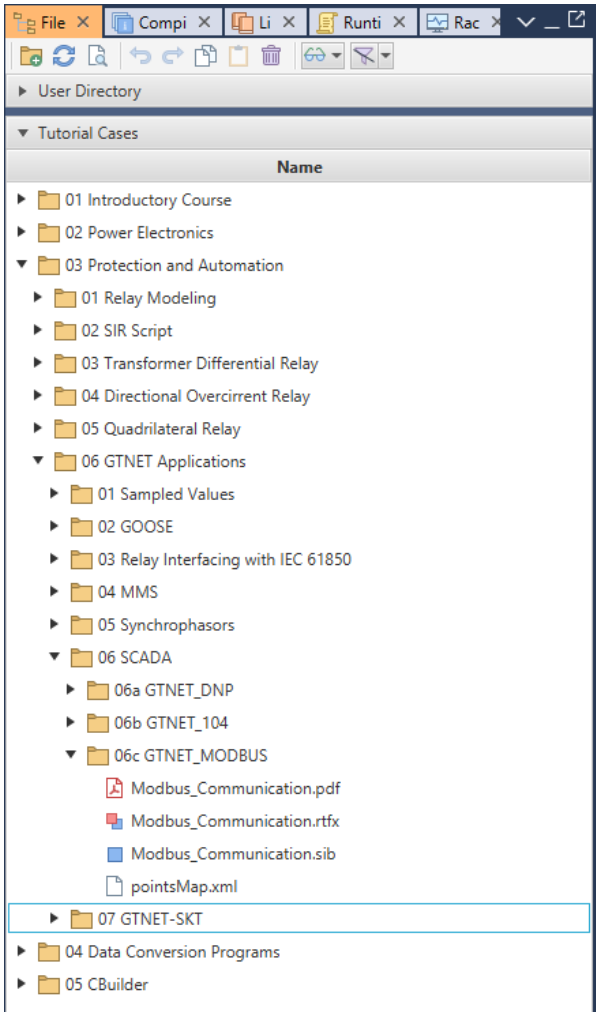
6 USING NS-3 TO MODIFY MODBUS PACKETS

Here, we use NS-3 to modify the MODBUS packets that goes through the NS-3 virtual network. These modification are done using ARP spoofing attacks. The instructions for running the NS-3 simulations are similar to that of DNP3. Therefore, we will not explain it here. The only difference is in the entries added in `/etc/ns3/ns3.conf`. Which we will explain in the subsequent subsections.

6.1. Example scenario: Modbus_Communication



The above case can be found in the Tutorials tab in RSCAD FX 2.0 at the following location:



6.2. Stating the new values to be assigned for MODBUS

The modifications to the MODBUS data are included in a file "`ns3.conf`". This file should be located in `/etc/ns3` in the Linux machine. The format of the text are given below

```
protocol modbus
<function_code> <Index> <value>
```

Here the file parser uses the string "protocol Modbus" to identify that the data following that line are for modbus packet modification. The next lines contain information on what to modify.

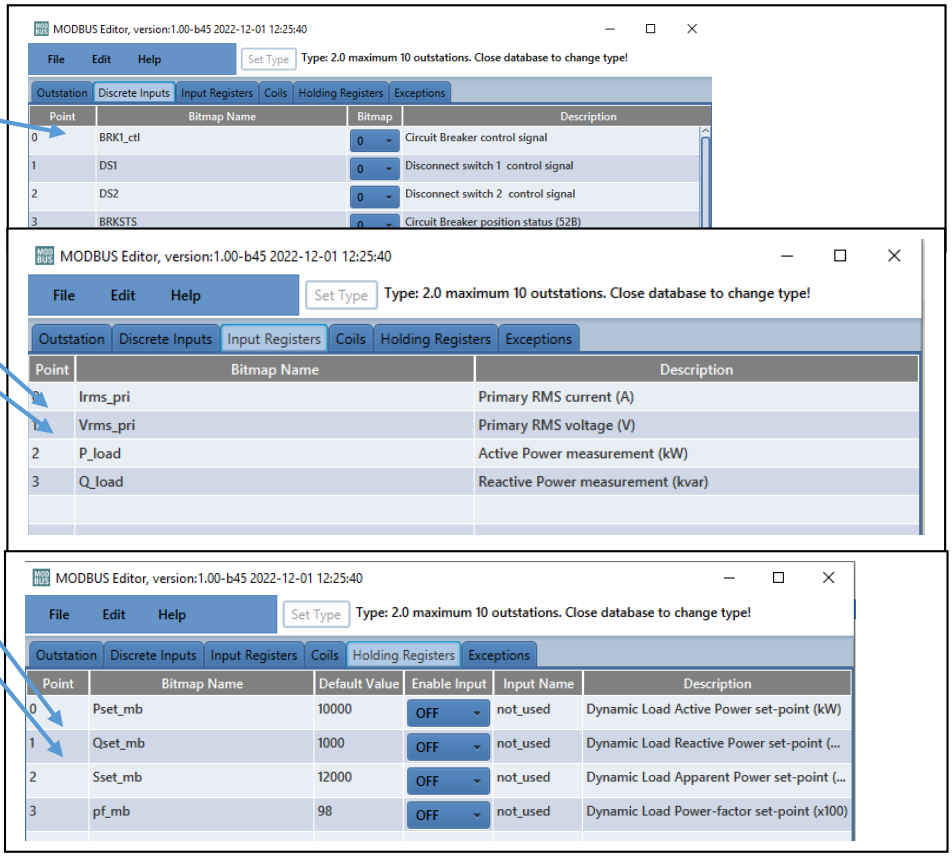
Function_code - The Function code of the modbus message of interest

Index - Is the index of the variable belonging to the stated Function code

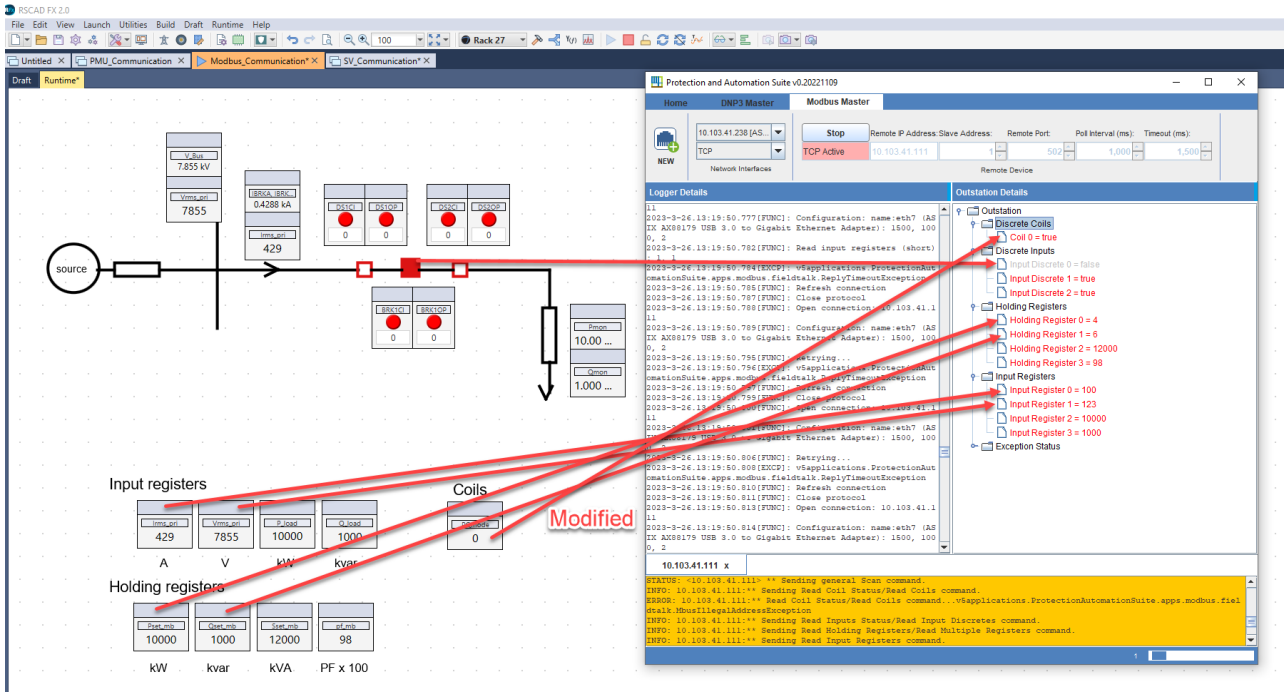
Value - Is the new parameter value to be assigned

An example is shown below:

```
protocol modbus
2 0 0
4 0 100
4 1 123
3 0 4
3 1 6
```



After running the simulation the following can be observed in the runtime.



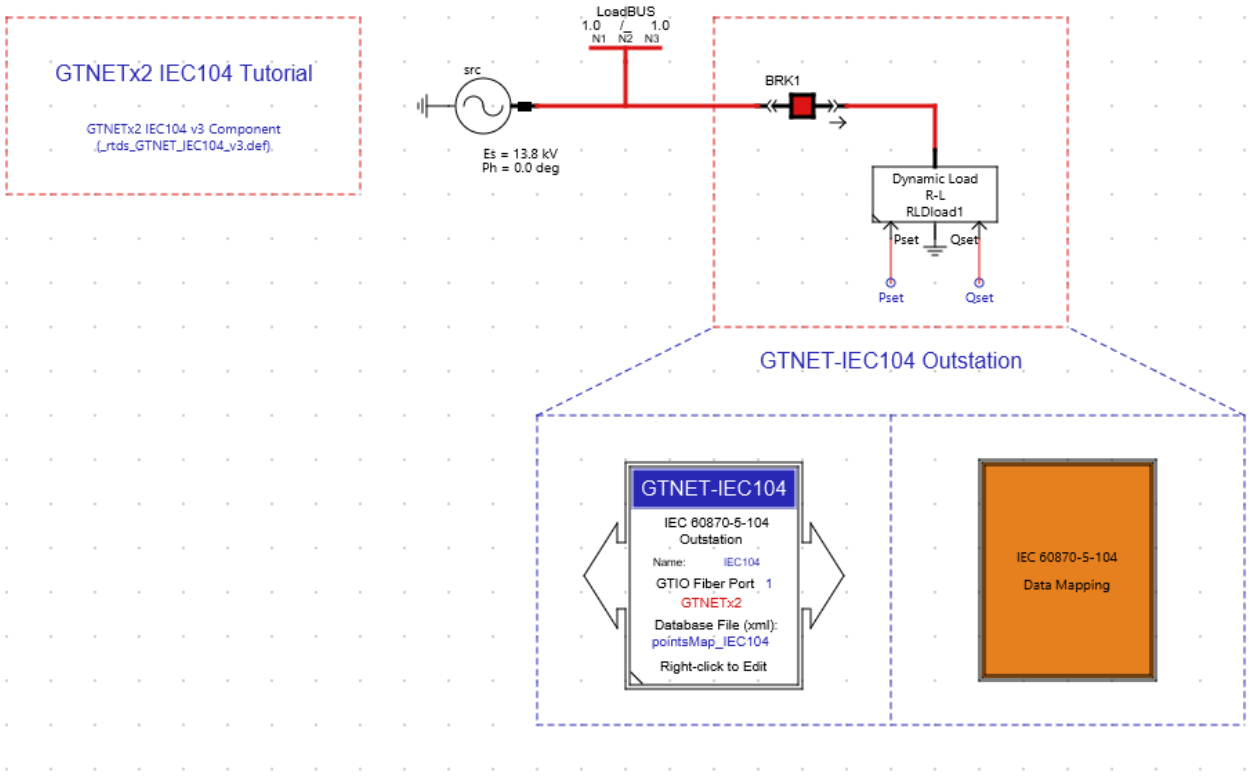
Now if we compare the wireshark captures on the server side and client side of these responses you would see that the packets have been modified. If you observe the destination Ethernet address of the responses at server side you will see that it is not the same as the clients. However in this scenario as all the nodes are in the same network it should be the same as the clients. That happens because of ARP spoofing and the Attacker node gets and sends the data running a Man-In-The-Middle attack. It is also the case at the Client side.

Server Side	Client side
Frame 7272537: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface enp4s0, id 0 Ethernet II, Src: RTDSTech_0b:5d (00:50:c2:4f:9b:5d), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04) Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170 Transmission Control Protocol, Src Port: 502, Dst Port: 17259, Seq: 1, Ack: 13, Len: 10 Modbus/TCP Modbus .000 0010 = Function Code: Read Discrete Inputs (2) [Request Frame: 7272533] [Time from request: 0.000386943 seconds] Byte Count: 1 Bit 0 : 1 Bit 1 : 1 Bit 2 : 1 Bit 3 : 0	Frame 2098229: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface enp5s0, id 0 Ethernet II, Src: 00:00:00_00:00:04 (00:00:00:00:00:04), Dst: Dell_05:d8:ff (10:65:30:05:d8:ff) Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170 Transmission Control Protocol, Src Port: 502, Dst Port: 17259, Seq: 1, Ack: 13, Len: 10 Modbus/TCP Modbus .000 0010 = Function Code: Read Discrete Inputs (2) [Request Frame: 2098225] [Time from request: 0.007384135 seconds] Byte Count: 1 Bit 0 : 0 Bit 1 : 1 Bit 2 : 1 Bit 3 : 0
Frame 7273667: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface enp4s0, id 0 Ethernet II, Src: RTDSTech_0b:5d (00:50:c2:4f:9b:5d), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04) Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170 Transmission Control Protocol, Src Port: 502, Dst Port: 17259, Seq: 11, Ack: 25, Len: 17 Modbus/TCP Modbus .000 0100 = Function Code: Read Input Registers (4) [Request Frame: 7273663] [Time from request: 0.000354602 seconds] Byte Count: 8 Register 0 (UINT16): 429 Register 1 (UINT16): 7855 Register 2 (UINT16): 10000 Register 3 (UINT16): 1000	Frame 2099356: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface enp5s0, id 0 Ethernet II, Src: 00:00:00_00:00:04 (00:00:00:00:00:04), Dst: Dell_05:d8:ff (10:65:30:05:d8:ff) Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170 Transmission Control Protocol, Src Port: 502, Dst Port: 17259, Seq: 11, Ack: 25, Len: 17 Modbus/TCP Modbus .000 0100 = Function Code: Read Input Registers (4) [Request Frame: 2099351] [Time from request: 0.007446524 seconds] Byte Count: 8 Register 0 (UINT16): 100 Register 1 (UINT16): 123 Register 2 (UINT16): 10000 Register 3 (UINT16): 1000
Frame 7274587: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface enp4s0, id 0 Ethernet II, Src: RTDSTech_0b:5d (00:50:c2:4f:9b:5d), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04) Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170 Transmission Control Protocol, Src Port: 502, Dst Port: 17259, Seq: 28, Ack: 37, Len: 17 Modbus/TCP Modbus .000 0011 = Function Code: Read Holding Registers (3) [Request Frame: 7274583] [Time from request: 0.000373893 seconds] Byte Count: 8 Register 0 (UINT16): 10000 Register 1 (UINT16): 1000 Register 2 (UINT16): 12000 Register 3 (UINT16): 98	Frame 2100274: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface enp5s0, id 0 Ethernet II, Src: 00:00:00_00:00:04 (00:00:00:00:00:04), Dst: Dell_05:d8:ff (10:65:30:05:d8:ff) Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170 Transmission Control Protocol, Src Port: 502, Dst Port: 17259, Seq: 28, Ack: 37, Len: 17 Modbus/TCP Modbus .000 0011 = Function Code: Read Holding Registers (3) [Request Frame: 2100269] [Time from request: 0.006904383 seconds] Byte Count: 8 Register 0 (UINT16): 4 Register 1 (UINT16): 6 Register 2 (UINT16): 12000 Register 3 (UINT16): 98

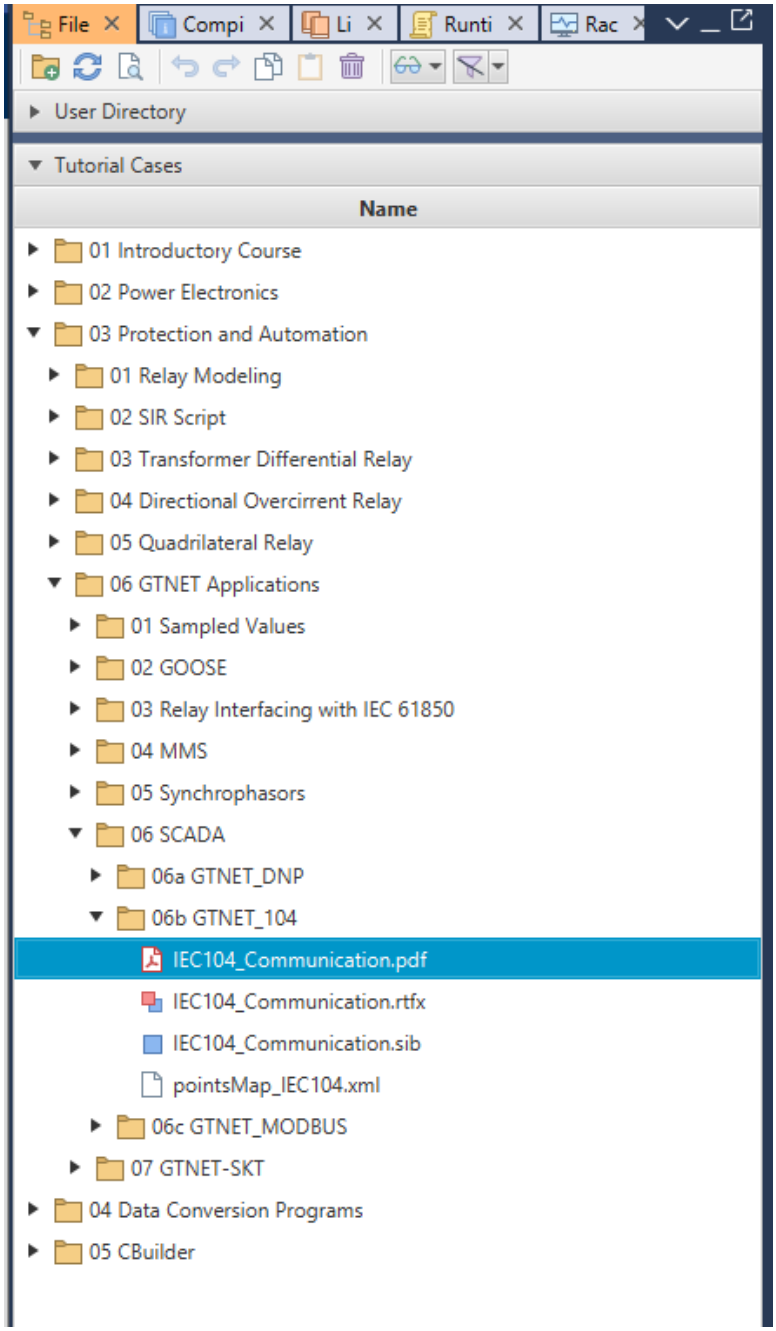
7 USING NS-3 TO MODIFY IEC104 PACKETS

Here, we use NS-3 to modify the IEC104 packets that goes through the NS-3 virtual network. These modification are done using ARP spoofing attacks. The instructions for running the NS-3 simulations are similar to that of DNP3. Therefore, we will not explain it here. The only difference is in the entries added in `/etc/ns3/ns3.conf`. Which we will explain in the subsequent subsections.

7.1. Example scenario: IEC104_Communication



The above case can be found in the Tutorials tab in RSCAD FX 2.0 at the following location:


File
Compi
Li
Runti
Rac
User Directory
Tutorial Cases
Name
01 Introductory Course
02 Power Electronics
03 Protection and Automation
01 Relay Modeling
02 SIR Script
03 Transformer Differential Relay
04 Directional Overcirrent Relay
05 Quadrilateral Relay
06 GTNET Applications
01 Sampled Values
02 GOOSE
03 Relay Interfacing with IEC 61850
04 MMS
05 Synchrophasors
06 SCADA
06a GTNET_DNP
06b GTNET_104
IEC104_Communication.pdf
IEC104_Communication.rtfx
IEC104_Communication.sib
pointsMap_IEC104.xml
06c GTNET_MODBUS
07 GTNET-SKT
04 Data Conversion Programs
05 CBuilder

7.2. Stating the new values to be assigned for IEC104

The modifications to the IEC104 data are included in a file "ns3.conf". This file should be located in /etc/ns3 in the Linux machine. The format of the text are given below

```
protocol iec104
<TypeID> <ASDU No> <IOA> <value>
```

Here the file parser uses the string "protocol iec104" to identify that the data following that line are for IEC104 packet modification. The next lines contain information on what to modify.

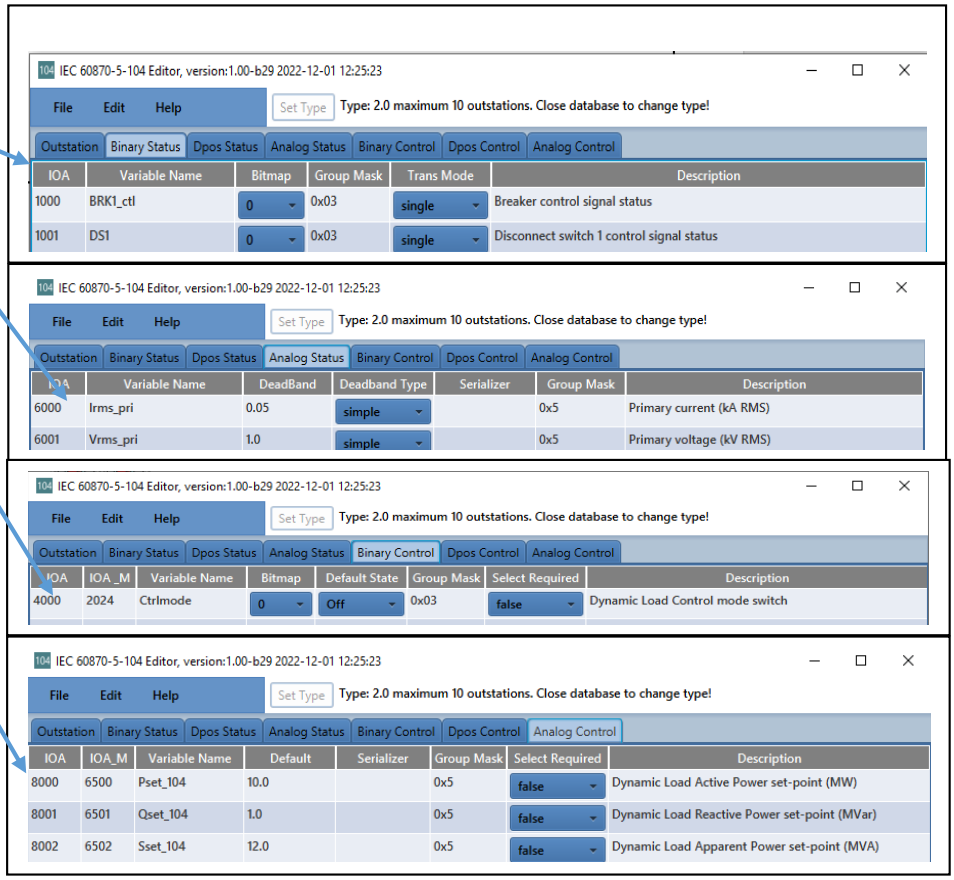
TypeID - The Function code of the IEC104 message of interest

ASDU No - This is the common address and it is association with all objects with in the ASDU
IOA- This is the Information object address. It identifies a particular data with in a station

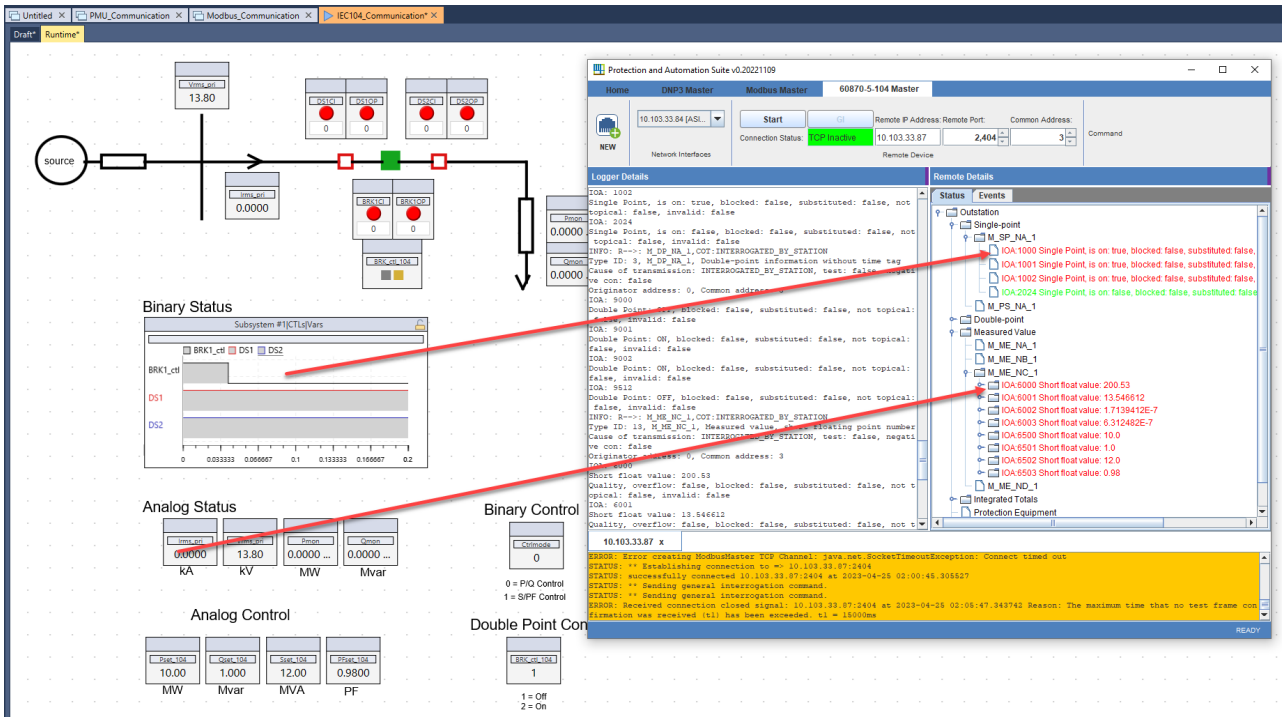
Value - Is the new parameter value to be assigned

An example is shown below:

```
protocol iec104
1 3 1000 1
13 3 6000 200.53
45 3 4000 0
50 3 8000 123.456
```



After running the simulation, the following can be observed in the runtime.



Now if we compare the wireshark captures on the server side and client side of these responses you would see that the packets have been modified. If you observe the destination Ethernet address of the responses at server side you will see that it is not the same as the clients. However in this scenario as all the nodes are in the same network it should be the same as the clients. That happens because of ARP spoofing and the Attacker node gets and sends the data running a Man-In-The-Middle attack. It is also the case at the Client side.

Server Side

```
Frame 428: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface enp4s0, id 0
Ethernet II, Src: RTDSTech_0b:5d (00:50:c2:4f:9b:5d), Dst: 00:00:00_00:00:04 (00:00:00:00:00:0
Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170
Transmission Control Protocol, Src Port: 2404, Dst Port: 31421, Seq: 17, Ack: 17, Len: 28
IEC 60870-5-104: -> I (33,5)
IEC 60870-5-101/104 ASDU: ASDU=3 M_SP_NA_1 Inrogen IOA[4]=1000,... 'single-point information'
    TypeId: M_SP_NA_1 (1)
    0... .... = SQ: False
    .000 0100 = NumIx: 4
    ..01 0100 = CauseTx: Inrogen (20)
    .0.. .... = Negative: False
    0... .... = Test: False
    OA: 0
    Addr: 3
  IOA: 1000
      IOA: 1000
    SIQ: 0x00
  IOA: 1001
  IOA: 1002
  IOA: 2024
```

```
Frame 430: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface enp4s0, id 0
Ethernet II, Src: RTDSTech_0b:5d (00:50:c2:4f:9b:5d), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170
Transmission Control Protocol, Src Port: 2404, Dst Port: 31421, Seq: 73, Ack: 17, Len: 76
IEC 60870-5-104: -> I (35,5)
IEC 60870-5-101/104 ASDU: ASDU=3 M_ME_NC_1 Inrogen IOA[8]=6000,... 'measured value, short floating point number
    TypeId: M_ME_NC_1 (13)
    0... .... = SQ: False
    .000 1000 = NumIx: 8
    ..01 0100 = CauseTx: Inrogen (20)
    .0.. .... = Negative: False
    0... .... = Test: False
    OA: 0
    Addr: 3
  IOA: 6000
      IOA: 6000
      Value: 9.12225e-08
    QDS: 0x00
  IOA: 6001
```

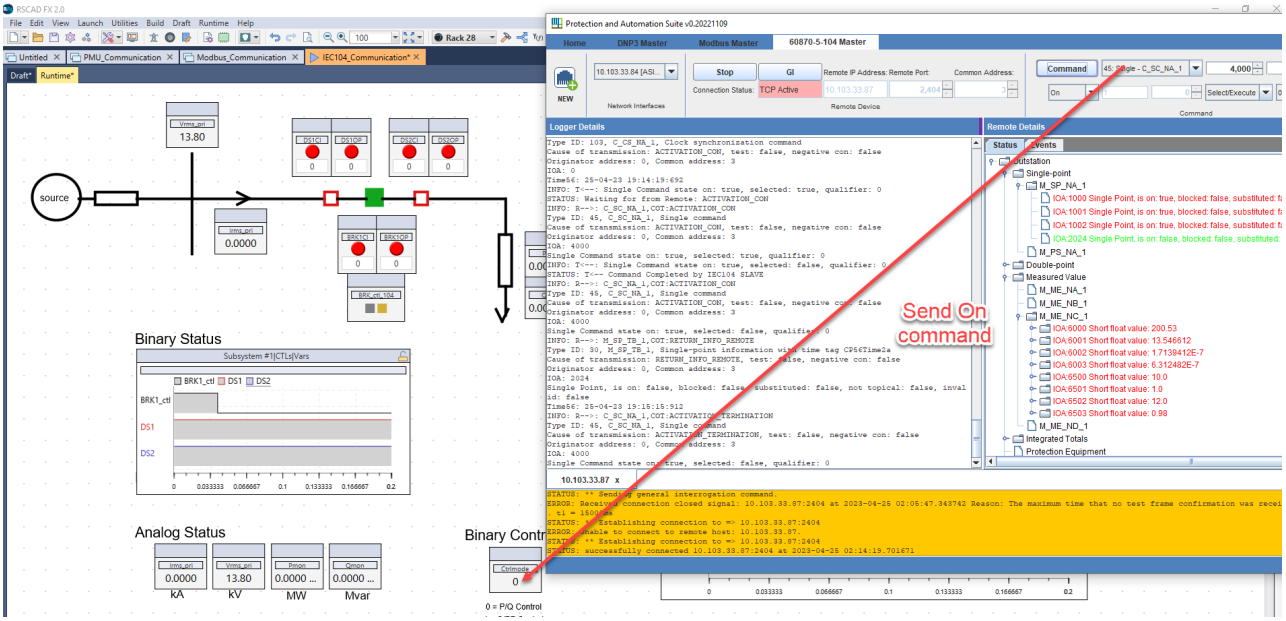
Client side

```
Frame 1115: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface enp5s0, id 0
Ethernet II, Src: 00:00:00_00:00:04 (00:00:00:00:00:04), Dst: Dell_05:d8:ff (10:65:30:05:d8:ff)
Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170
Transmission Control Protocol, Src Port: 2404, Dst Port: 31421, Seq: 23, Ack: 23, Len: 28
IEC 60870-5-104: -> I (33,5)
IEC 60870-5-101/104 ASDU: ASDU=3 M_SP_NA_1 Inrogen IOA[4]=1000,... 'single-point information'
    TypeId: M_SP_NA_1 (1)
    0... .... = SQ: False
    .000 0100 = NumIx: 4
    ..01 0100 = CauseTx: Inrogen (20)
    .0.. .... = Negative: False
    0... .... = Test: False
    OA: 0
    Addr: 3
  IOA: 1000
      IOA: 1000
    SIQ: 0x01
  IOA: 1001
  IOA: 1002
  IOA: 2024
```

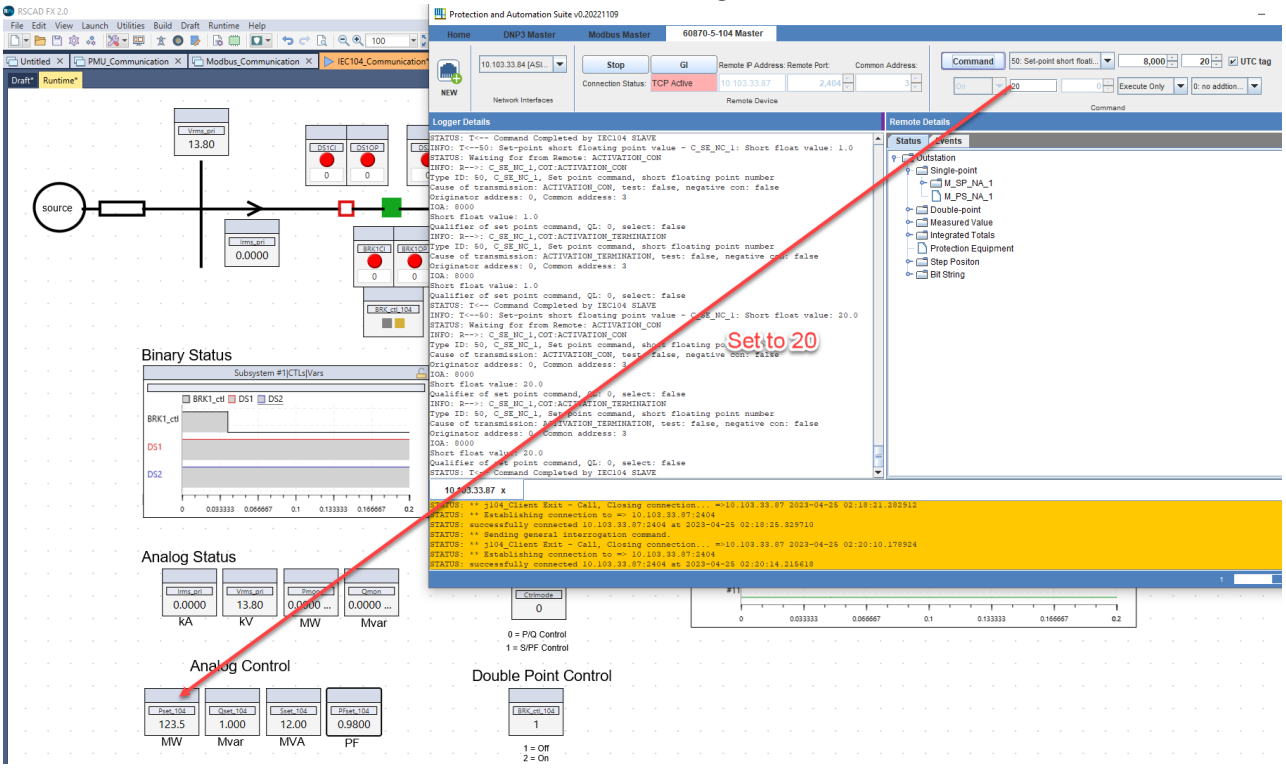
```
Frame 1118: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface enp5s0, id 0
Ethernet II, Src: 00:00:00_00:00:04 (00:00:00:00:00:04), Dst: Dell_05:d8:ff (10:65:30:05:d8:ff)
Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170
Transmission Control Protocol, Src Port: 2404, Dst Port: 31421, Seq: 79, Ack: 23, Len: 76
IEC 60870-5-104: -> I (35,5)
IEC 60870-5-101/104 ASDU: ASDU=3 M_ME_NC_1 Inrogen IOA[8]=6000,... 'measured value, short floating point numbe
    TypeId: M_ME_NC_1 (13)
    0... .... = SQ: False
    .000 1000 = NumIx: 8
    ..01 0100 = CauseTx: Inrogen (20)
    .0.. .... = Negative: False
    0... .... = Test: False
    OA: 0
    Addr: 3
  IOA: 6000
      IOA: 6000
      Value: 200.53
    QDS: 0x00
  IOA: 6001
```

A modification to Binary control (IOA 4000) command is done on line 3 of "ns3.conf". When we try to

change the value of CtrlMode from 0 to 1, it does not carry out in the runtime because of this modification. Therefore, the CtrlMode remains in state 0 even after the execution of the command.



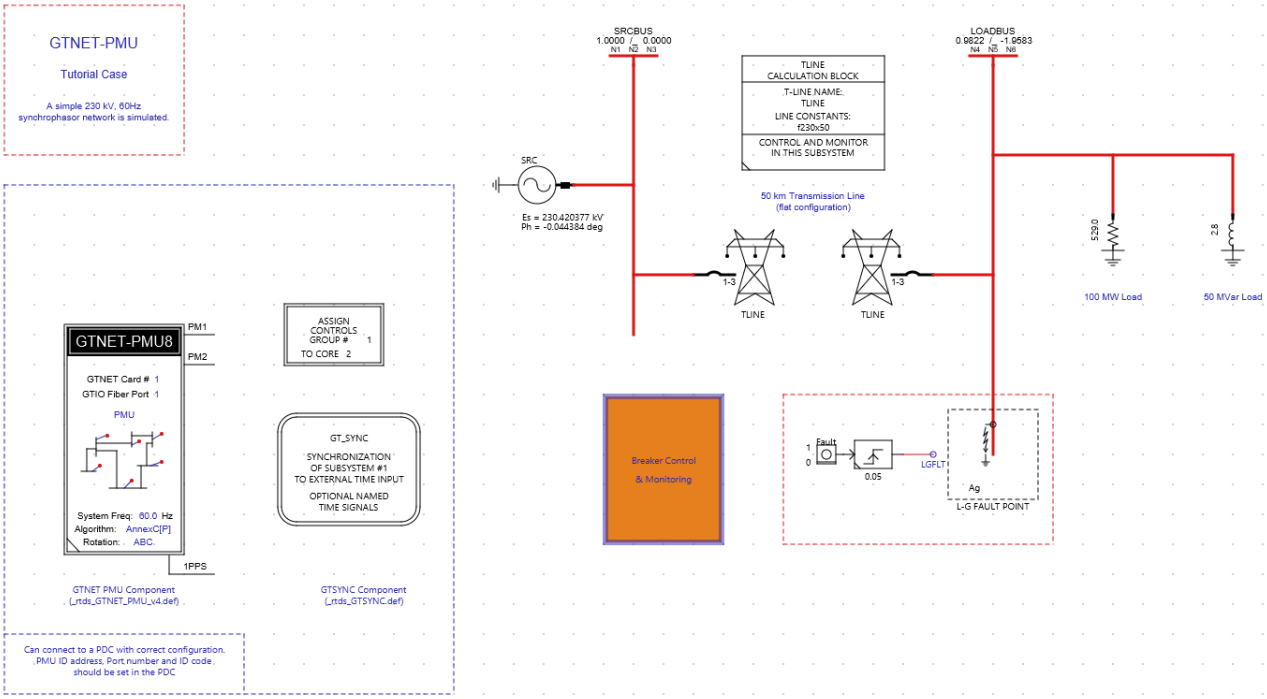
A modification to Analog control (IOA 8000) command is done on line 4 of "ns3.conf". When we try to set the value of Pset_104 from 10 to 20, it sets the value to 123.5 in the runtime because of this modification. The value becomes 123.5 instead of 123.456 due to rounding off.



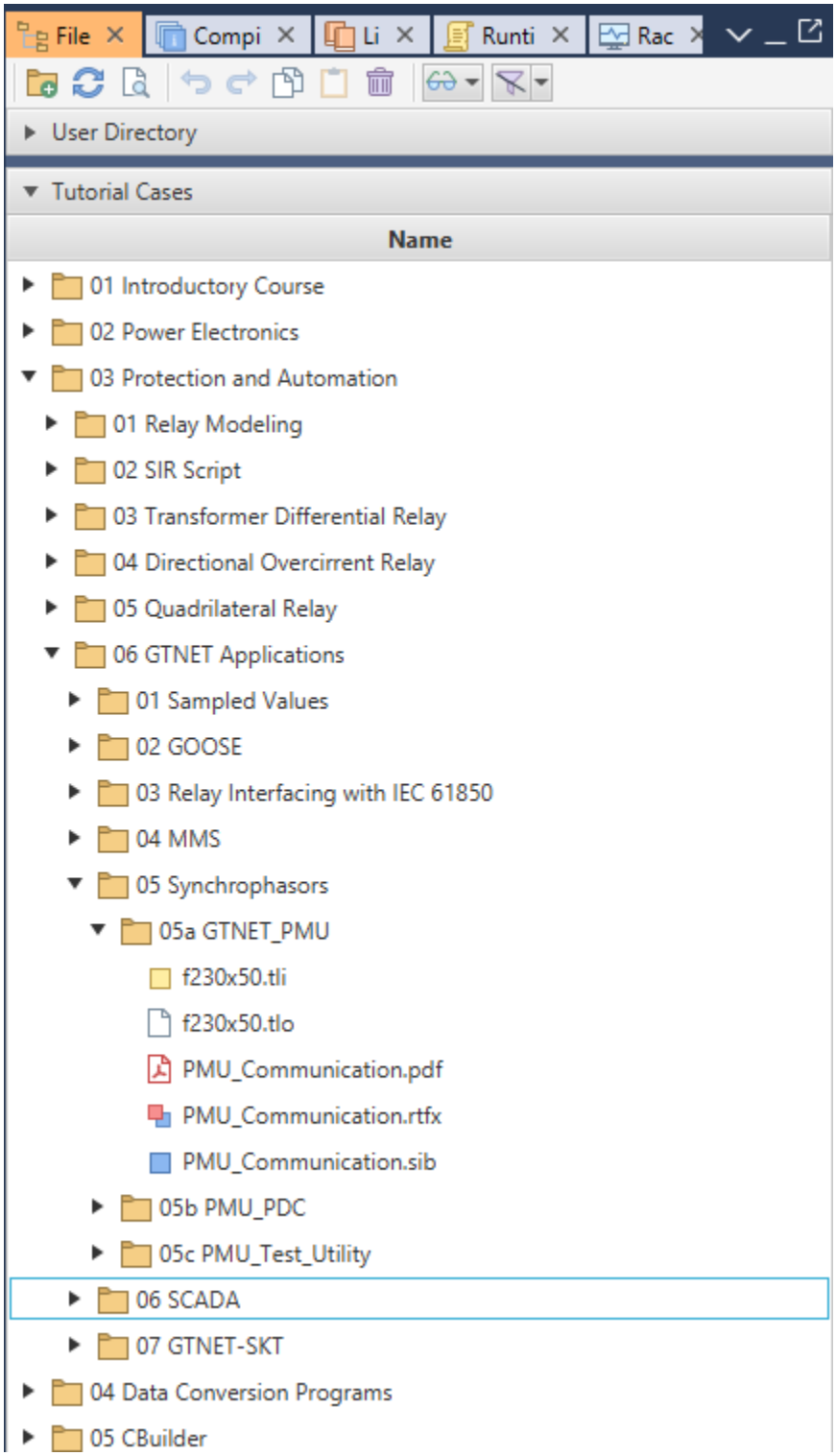
8 USING NS-3 TO MODIFY PMU PACKETS

Here, we use NS-3 to modify the Synchro-phasor packets that goes through the NS-3 virtual network. These modification are done using ARP spoofing attacks. The instructions for running the NS-3 simulations are similar to that of DNP3. Therefore, we will not explain it here. The only difference is in the entries added in `/etc/ns3/ns3.conf`. Which we will explain in the subsequent subsections.

8.1. Example scenario: PMU_Communication



The above case can be found in the Tutorials tab in RSCAD FX 2.0 at the following location:


File
Compi
Li
Runti
Rac
User Directory
Tutorial Cases
Name
01 Introductory Course
02 Power Electronics
03 Protection and Automation
01 Relay Modeling
02 SIR Script
03 Transformer Differential Relay
04 Directional Overcirrent Relay
05 Quadrilateral Relay
06 GTNET Applications
01 Sampled Values
02 GOOSE
03 Relay Interfacing with IEC 61850
04 MMS
05 Synchrophasors
05a GTNET_PMU
f230x50.tli
f230x50.tlo
PMU_Communication.pdf
PMU_Communication.rtfx
PMU_Communication.sib
05b PMU_PDC
05c PMU_Test_Utility
06 SCADA
07 GTNET-SKT
04 Data Conversion Programs
05 CBuilder

8.2. Stating the new values to be assigned for PMU

The modifications to the PMU data are included in a file "ns3.conf". This file should be located in /etc/ns3 in the Linux machine. The format of the text are given below

```
protocol pmu
<Station> <DataType> <Data name> <value>
```

Here the file parser uses the string "protocol pmu" to identify that the data following that line are for PMU packet modification. The next lines contain information on what to modify.

Station - The name of the station

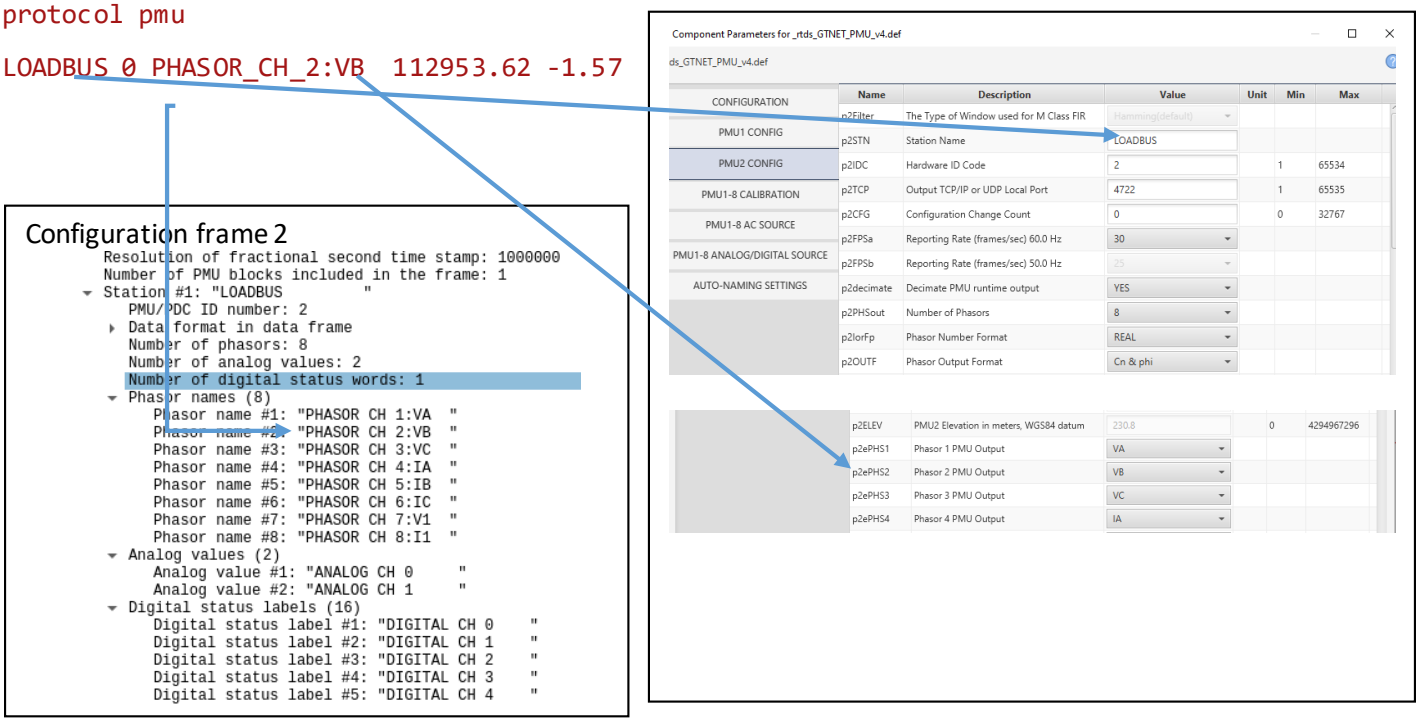
DataType - This states if the data are Phasors (DataType=0), Analog values (DataType=1), Digital values (DataType=2)

Data name - This is the name of the data appearing in the Configuration frame 2. Here the spaces are replaced with underscores.

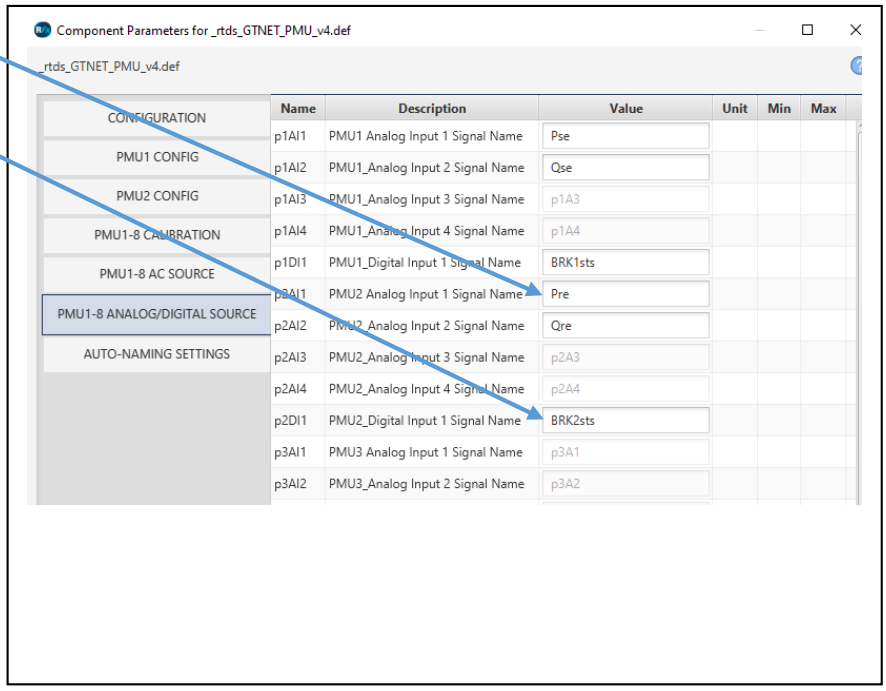
Value - Is the new parameter value to be assigned. In the case of Phasors it will be a pair of values. You should know if the data are transmitted using Cartesian or polar format and put the values accordingly. The angles should be specified in radians not degrees.

An example is shown below:

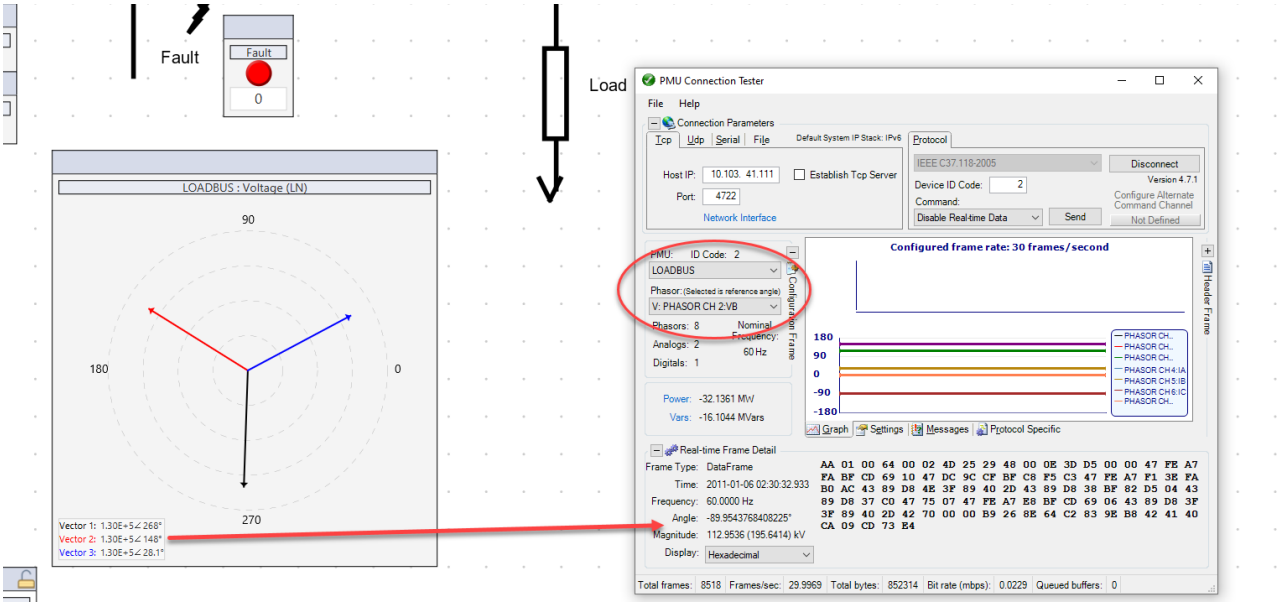
```
protocol pmu
LOADBUS 0 PHASOR_CH_2:VB  112953.62 -1.57
```



```
LOADBUS 1 ANALOG_CH_0 100.0
LOADBUS 2 DIGITAL_CH_0 0
```



Here we use the software called PMU connection tester to monitor the phasor data. After running the simulation, the following can be observed in the runtime and the PMU connection tester.



Now if we compare the wireshark captures on the server side and client side of these responses you would see that the packets have been modified. If you observe the destination Ethernet address of the responses at server side you will see that it is not the same as the clients. However,

in this scenario as all the nodes are in the same network it should be the same as the clients. That happens because of ARP spoofing and the Attacker node gets and sends the data running a Man-In-The-Middle attack. It is also the case at the Client side.

Server Side

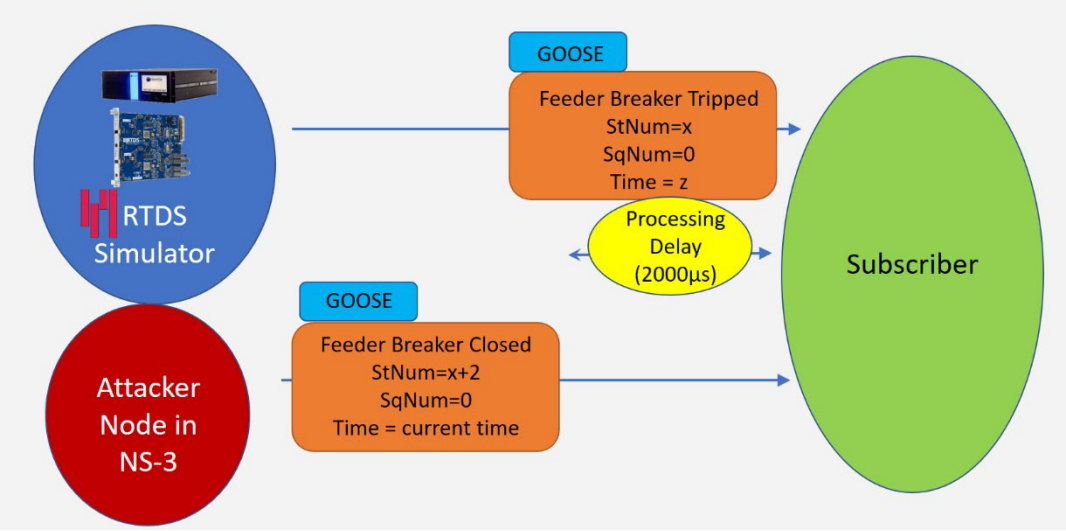
```
▸ Frame 551752: 154 bytes on wire (1232 bits), 154 bytes captured (1232 bits) on interface enp4s0, id 0
▸ Ethernet II, Src: RTDSTech_0b:5d (00:50:c2:4f:9b:5d), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
▸ Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170
▸ Transmission Control Protocol, Src Port: 4722, Dst Port: 34324, Seq: 515, Ack: 55, Len: 100
▾ IEEE C37.118 Synchrophasor Protocol, Data Frame [correct]
  ▸ Synchronization word: 0xaa01
    Framesize: 100
    PMU/PDC ID number: 2
    SOC time stamp: Mar 17, 2023 16:13:11.000000000 UTC
  ▸ Time quality flags
    Fraction of second (raw): 833333
  ▾ Measurement data, using frame number 551744 as configuration frame
    ▾ Station: "LOADBUS         "
      ▸ Flags
      ▾ Phasors (8)
          Phasor #1: "PHASOR CH 1:VA  ",  130383.95V/_ -91.95°
          Phasor #2: "PHASOR CH 2:VB  ",  130383.60V/_ 148.05°
          Phasor #3: "PHASOR CH 3:VC  ",  130383.88V/_  28.05°
          Phasor #4: "PHASOR CH 4:IA  ",     275.69A/_  61.44°
          Phasor #5: "PHASOR CH 5:IB  ",     275.69A/_ -58.56°
          Phasor #6: "PHASOR CH 6:IC  ",     275.69A/_-178.56°
          Phasor #7: "PHASOR CH 7:V1  ",  130383.81V/_ -91.95°
          Phasor #8: "PHASOR CH 8:I1  ",     275.69A/_  61.44°
        Actual frequency value: 60Hz
        Rate of change of frequency: -0.000159097Hz/s
      ▾ Analog values (2)
          Analog value #1: "ANALOG CH 0     ", 96.431
          Analog value #2: "ANALOG CH 1     ", 48.313
      ▾ Digital status words (1)
          Digital status word #1: 0x0001
```

Client side

```
▸ Frame 439447: 154 bytes on wire (1232 bits), 154 bytes captured (1232 bits) on interface enp5s0, id 0
▸ Ethernet II, Src: 00:00:00_00:00:04 (00:00:00:00:00:04), Dst: Dell_05:d8:ff (10:65:30:05:d8:ff)
▸ Internet Protocol Version 4, Src: 172.24.50.91, Dst: 172.24.50.170
▸ Transmission Control Protocol, Src Port: 4722, Dst Port: 34740, Seq: 966515, Ack: 55, Len: 100
▾ IEEE C37.118 Synchrophasor Protocol, Data Frame [correct]
  ▸ Synchronization word: 0xaa01
    Framesize: 100
    PMU/PDC ID number: 2
    SOC time stamp: Mar 17, 2023 16:28:41.000000000 UTC
  ▸ Time quality flags
    Fraction of second (raw): 766667
  ▾ Measurement data, using frame number 408068 as configuration frame
    ▾ Station: "LOADBUS         "
      ▸ Flags
      ▾ Phasors (8)
          Phasor #1: "PHASOR CH 1:VA  ",  130383.90V/_ -91.95°
          Phasor #2: "PHASOR CH 2:VB  ",  112953.62V/_ -89.95°
          Phasor #3: "PHASOR CH 3:VC  ",  130383.59V/_  28.05°
          Phasor #4: "PHASOR CH 4:IA  ",     275.69A/_  61.44°
          Phasor #5: "PHASOR CH 5:IB  ",     275.69A/_ -58.56°
          Phasor #6: "PHASOR CH 6:IC  ",     275.69A/_-178.56°
          Phasor #7: "PHASOR CH 7:V1  ",  130383.81V/_ -91.95°
          Phasor #8: "PHASOR CH 8:I1  ",     275.69A/_  61.44°
        Actual frequency value: 60Hz
        Rate of change of frequency: 6.92098e-05Hz/s
      ▾ Analog values (2)
          Analog value #1: "ANALOG CH 0     ", 100.000
          Analog value #2: "ANALOG CH 1     ", 48.313
      ▾ Digital status words (1)
          Digital status word #1: 0x0000
```

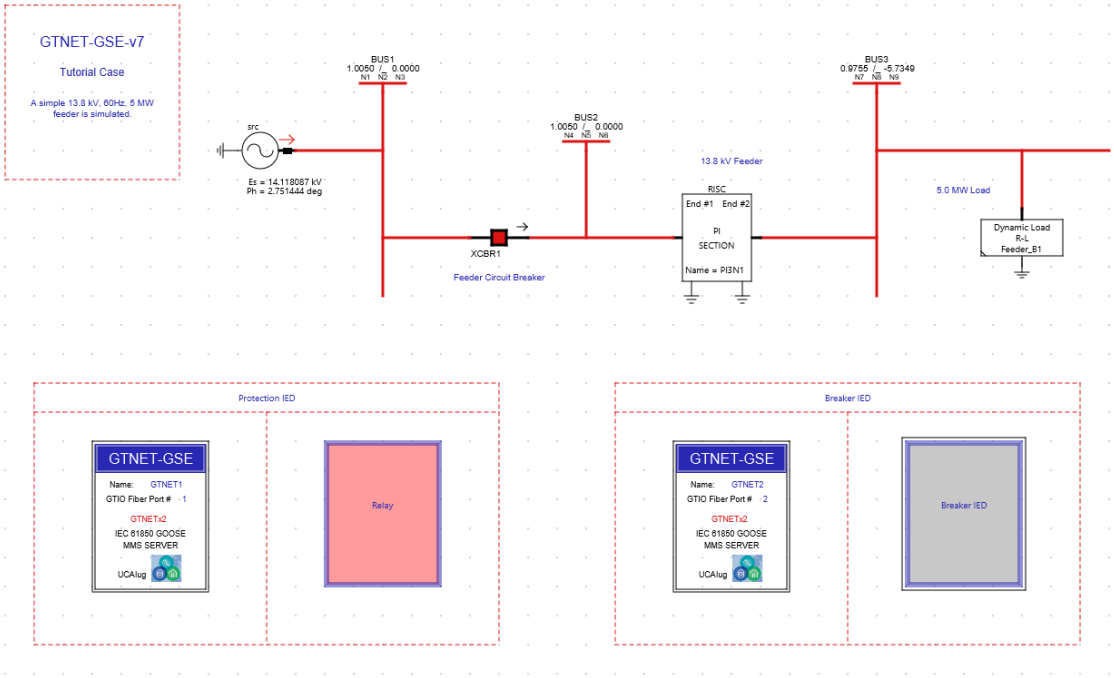
9 USING NS-3 TO MODIFY GOOSE Frames

Here, we use NS-3 to modify the GOOSE frames that arrives at the tap device of NS-3 virtual network. These modification are done by sending a malicious GOOSE frame. This GOOSE frame will increment the stNum field of the GOOSE frame and reset the sqNum field as shown in the figure below:

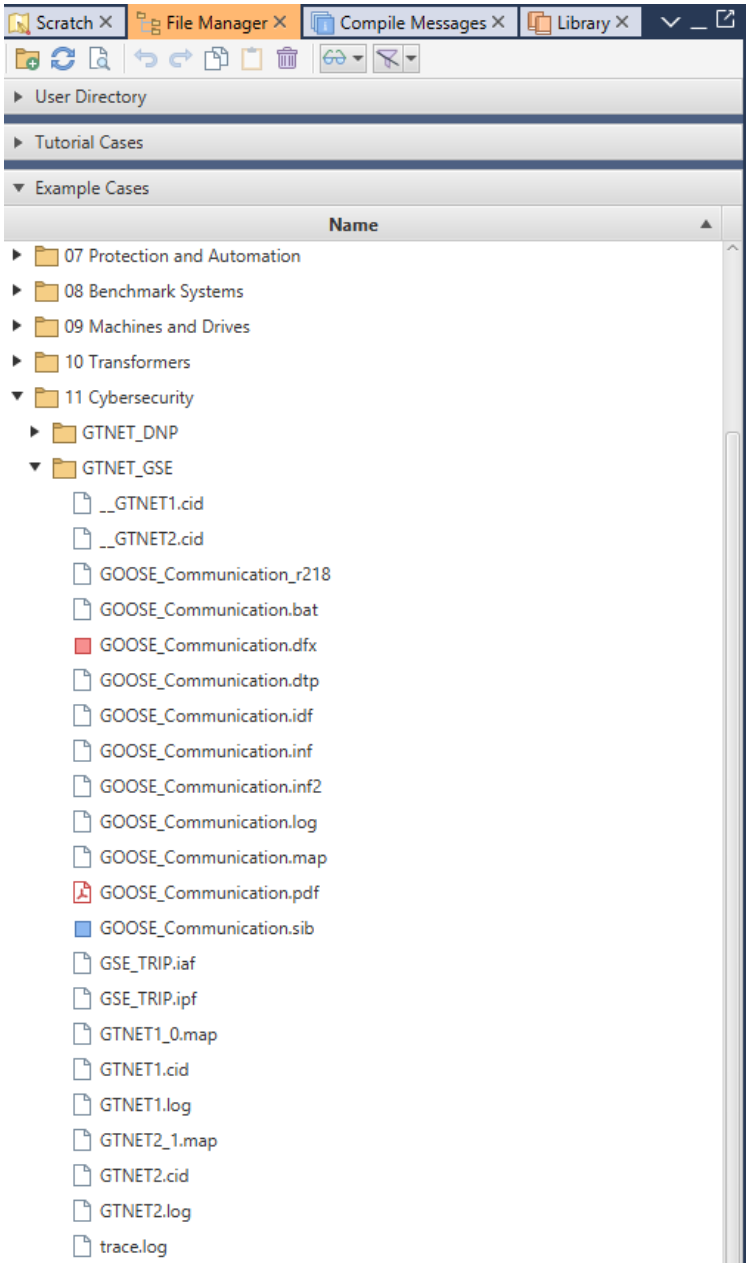


The instructions for running the NS-3 simulations are similar to that of DNP3 given in section 5.1. Therefore, we will not explain it here. The only difference is in the entries added in `/etc/ns3/ns3.conf`. Which we will explain in the subsequent subsections.

9.1. Example scenario: GOOSE_Communication



The above case can be found in the Tutorials tab in RSCAD FX 2.0 at the following location:


Scratch
File Manager
Compile Messages
Library
User Directory
Tutorial Cases
Example Cases
Name
07 Protection and Automation
08 Benchmark Systems
09 Machines and Drives
10 Transformers
11 Cybersecurity
GTNET_DNP
GTNET_GSE
__GTNET1.cid
__GTNET2.cid
GOOSE_Communication_r218
GOOSE_Communication.bat
GOOSE_Communication.dfx
GOOSE_Communication.dtp
GOOSE_Communication.idf
GOOSE_Communication.inf
GOOSE_Communication.inf2
GOOSE_Communication.log
GOOSE_Communication.map
GOOSE_Communication.pdf
GOOSE_Communication.sib
GSE_TRIP.iaf
GSE_TRIP.ipf
GTNET1_0.map
GTNET1.cid
GTNET1.log
GTNET2_1.map
GTNET2.cid
GTNET2.log
trace.log

9.2. Stating the new values to be assigned for GOOSE

The modifications to the GOOSE data are included in a file "ns3.conf". This file should be located in /etc/ns3 in the Linux machine. The format of the text are given below

```
protocol goose
<GocbRef> <DataSet> <DataIndex> <value>
```

Here the file parser uses the string "protocol goose" to identify that the data following that line are for GOOSE frame modification. The next lines contain information on what to modify.

GocbRef - The name of the goose control block

DataSet - The name of the data set

DataIndex - The index of the data within that data set

Value - Is the new parameter value to be assigned.

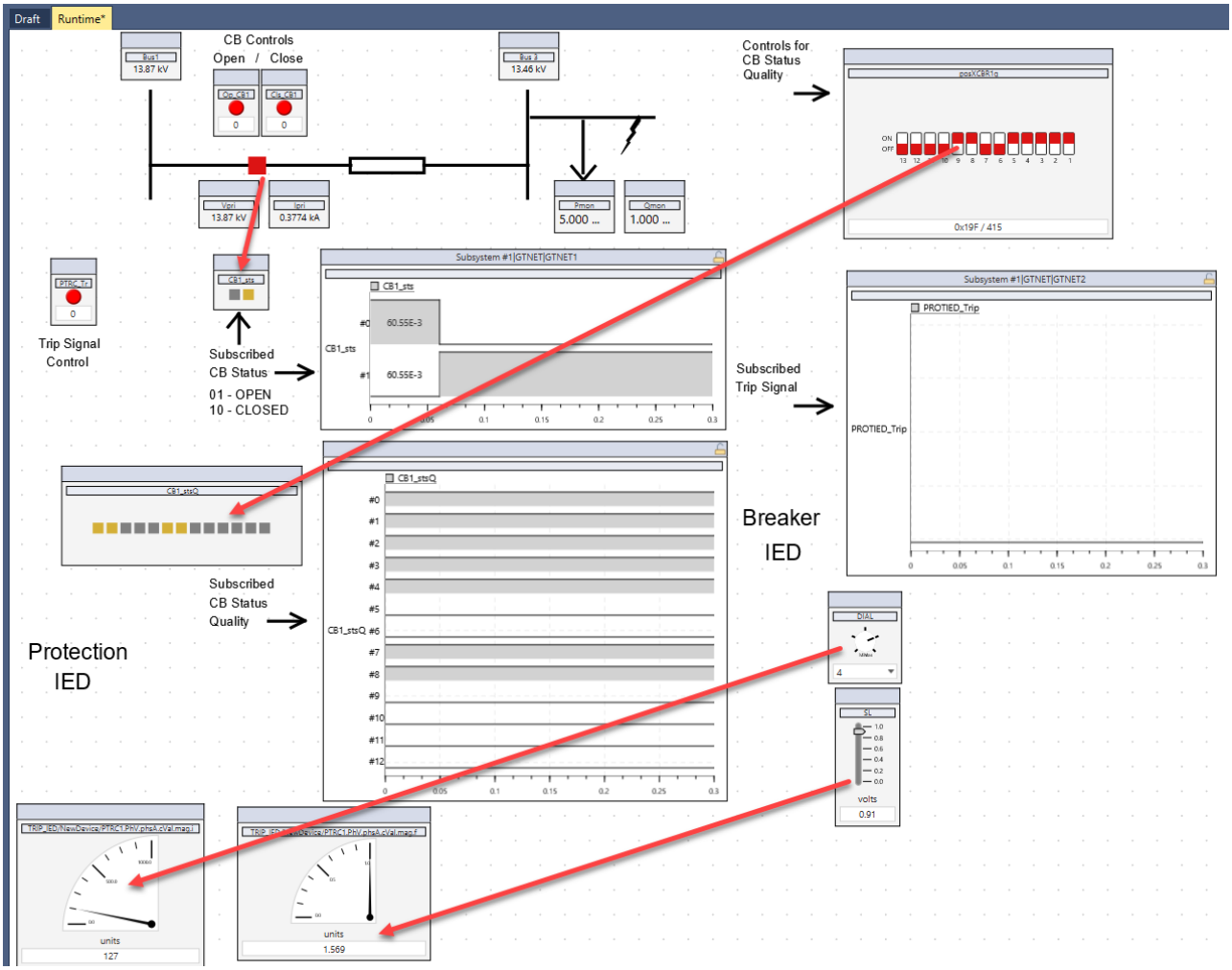
An example is shown below:

```
protocol goose
BRK_IEDNewDevice/LLN0$GO$BRK_IED  BRK_IEDNewDevice/LLN0$Goose_dataset2  0  0640  \
BRK_IEDNewDevice/LLN0$GO$BRK_IED  BRK_IEDNewDevice/LLN0$Goose_dataset2  1 030318  \
BRK_IEDNewDevice/LLN0$GO$BRK_IED  BRK_IEDNewDevice/LLN0$Goose_dataset2  2  127 \
BRK_IEDNewDevice/LLN0$GO$BRK_IED  BRK_IEDNewDevice/LLN0$Goose_dataset2  3  1.569
```

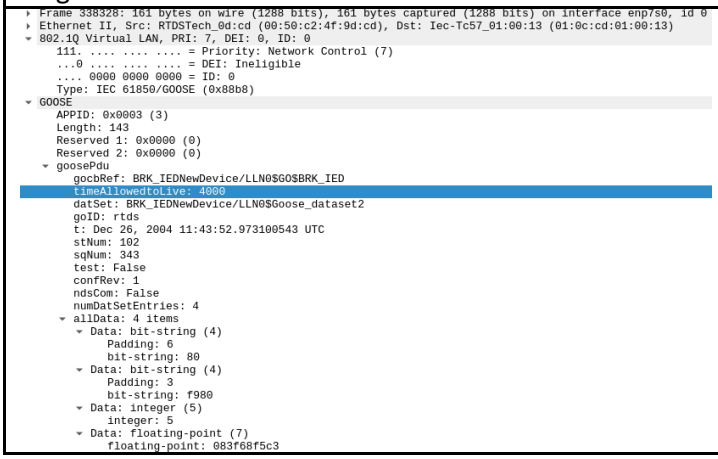
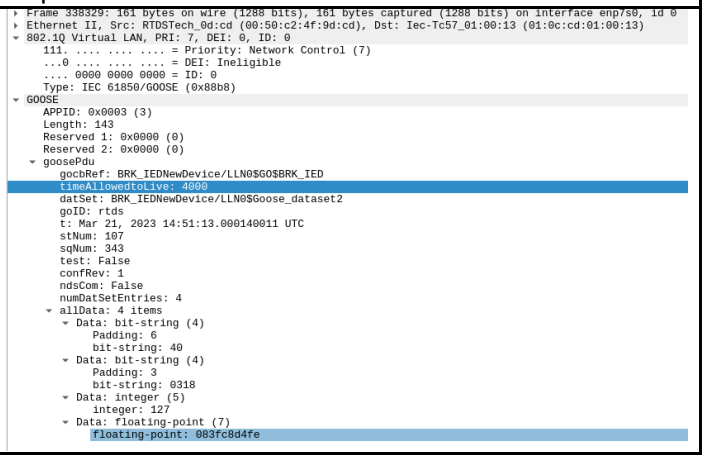
Here we are interested in the goose frames with the gocdRef, BRK_IEDNewDevice/LLN0GOBRK_IED and the dataset BRK_IEDNewDevice/LLN0$Goose_dataset2. These data frames are created by the case that we are running. Each of these data frames that we create in the GTNET card for the case mentioned above, will have 4 data fields. The first two are bit-string followed by an integer and a floating point value. The values for the bit strings consists of two parts and it is written as a hexadecimal. The first byte gives the number of padding bits at the end of the bit string. The second byte gives the value. For an example if we take the first value 0640, 06 is the number of padding bits and 40 is the value. If we write 0x40 in binary we get: b01000000. The last 6 bits are padding bits. Therefore, the values is b01. This value is the status of a breaker, b01 means the breaker is OPEN. Therefore, the subscriber will always see the breaker as OPEN immaterial of the actual state of the breaker. The second field in the goose frame of interest is the quality bit value of the breaker

status. It is also of bit-string type. We are modifying the values of it to `b0000001100011`. In the third line we are modifying the integer value of `127`. Finally we are modifying the floating point value to `1.569`.

Now lets see this modifications in action. Run the simulation file `rtds-Tap-ICS-Mod-One_Net` in NS-3 using the instructions provided in section 5.1. Then start the RSCAD simulation. After running the simulation, the following can be observed in the runtime. If you stop the NS-3 simulation at any point you will see that the actual value that matches the values at the BREAKER IED is shown at the TRIP IED.



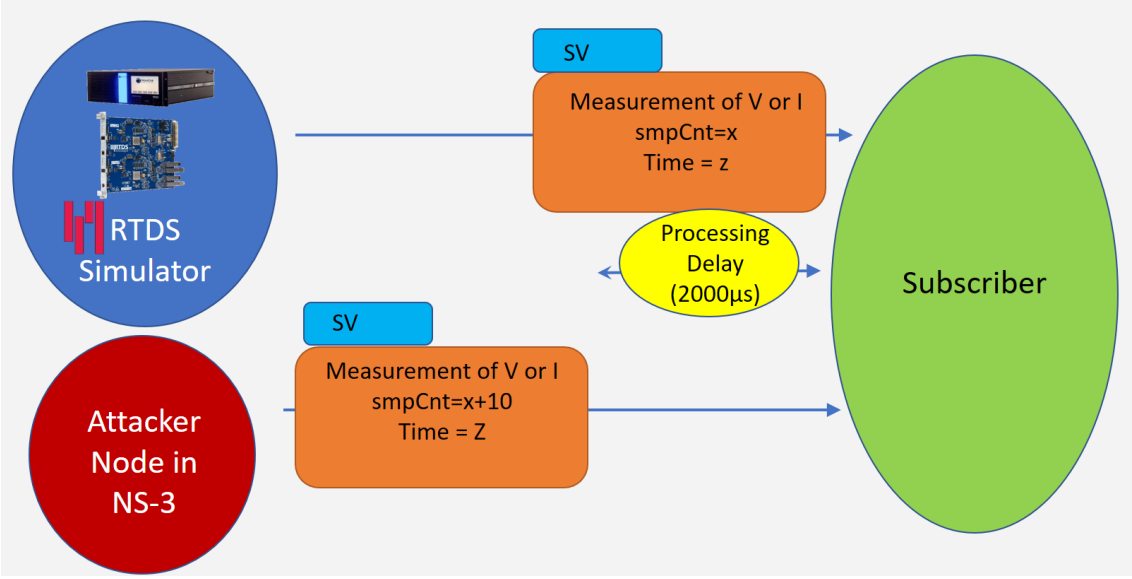
Now lets study the wireshark capture taken at any interface connected to the network the GTNET card is connected. One observation you can make is that there is a duplicate frame created for each Ethernet multicast frame the Breaker IED sends. Now lets compare those Ethernet frames.

Original	Duplicate
	

In additional to the changes in the data fields, you will see that the stNum field of the duplicate message is higher than that of the Original. This higher stNum ensures that this false goose frame gets accepted by the subscriber. Furthermore, we have modified the time field in the duplicate message to have the current time.

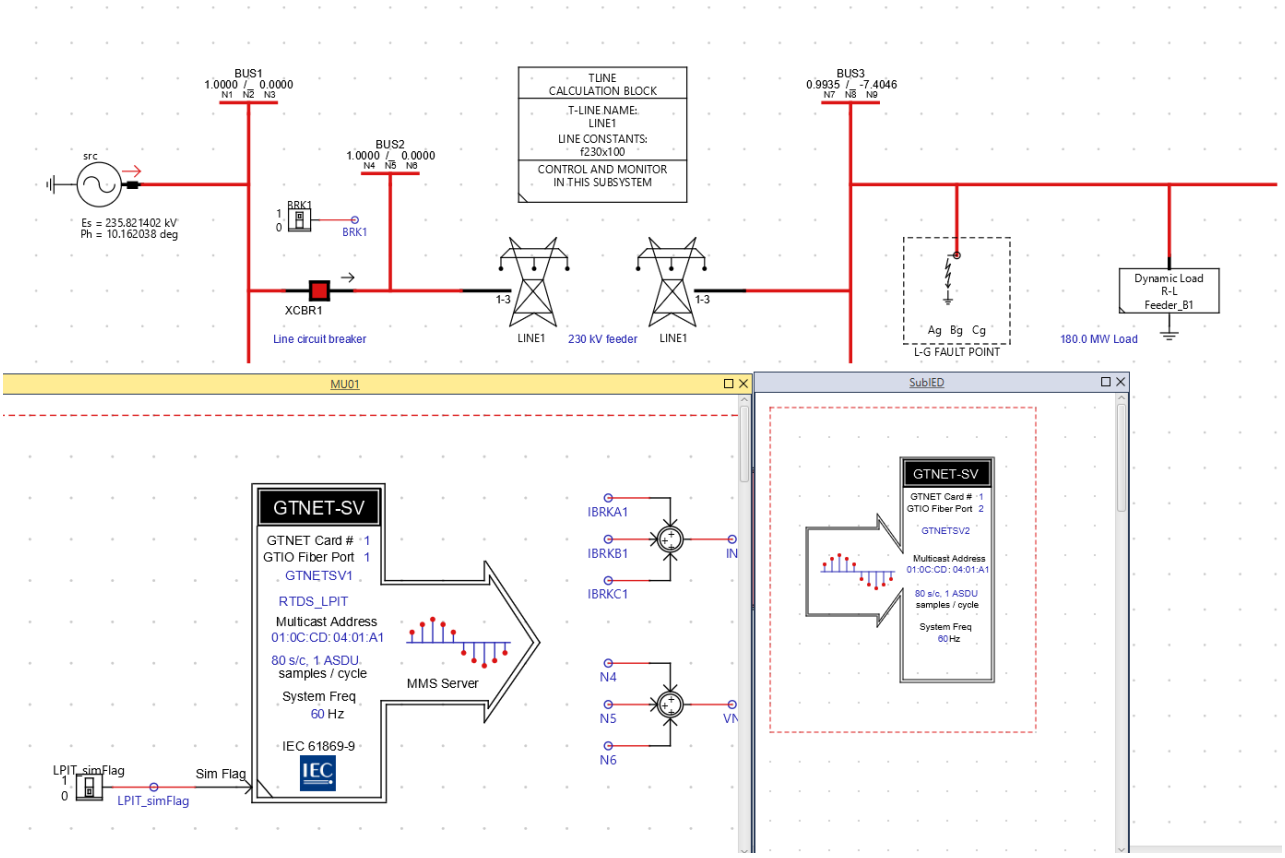
10 USING NS-3 TO MODIFY Sampled value Frames

Here, we use NS-3 to modify the Sampled Value frames that arrives at the tap device of NS-3 virtual network. These modification are done by sending a malicious SV frame. This SV frame will increment the smpCnt field of the SV frame field as shown in the figure below:

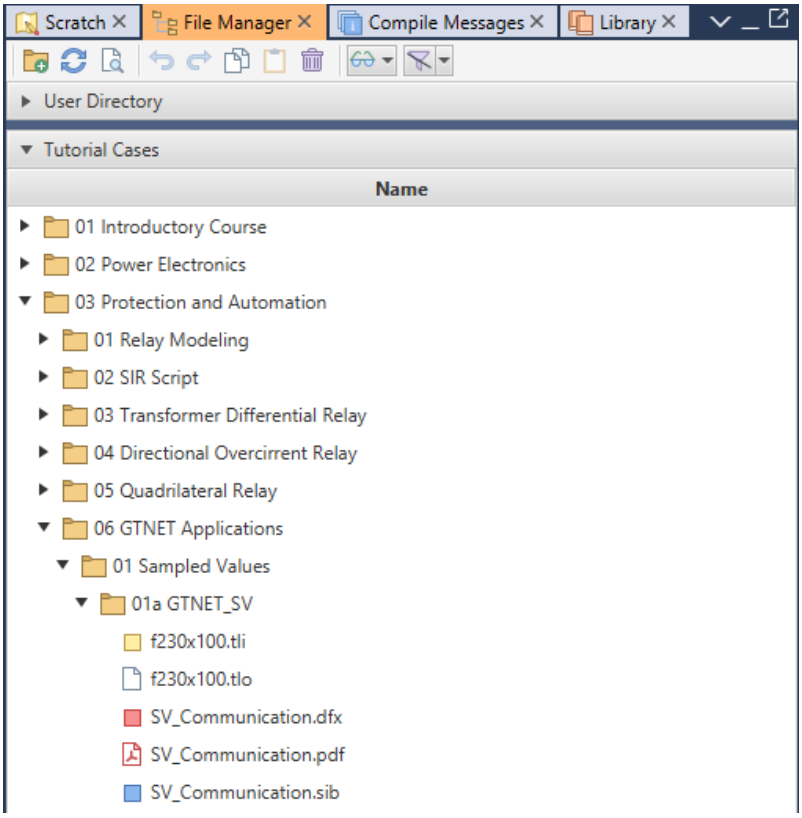


The instructions for running the NS-3 simulations are similar to that of DNP3 given in section 5.1. Therefore, we will not explain it here. The only difference is in the entries added in `/etc/ns3/ns3.conf`. Which we will explain in the subsequent subsections.

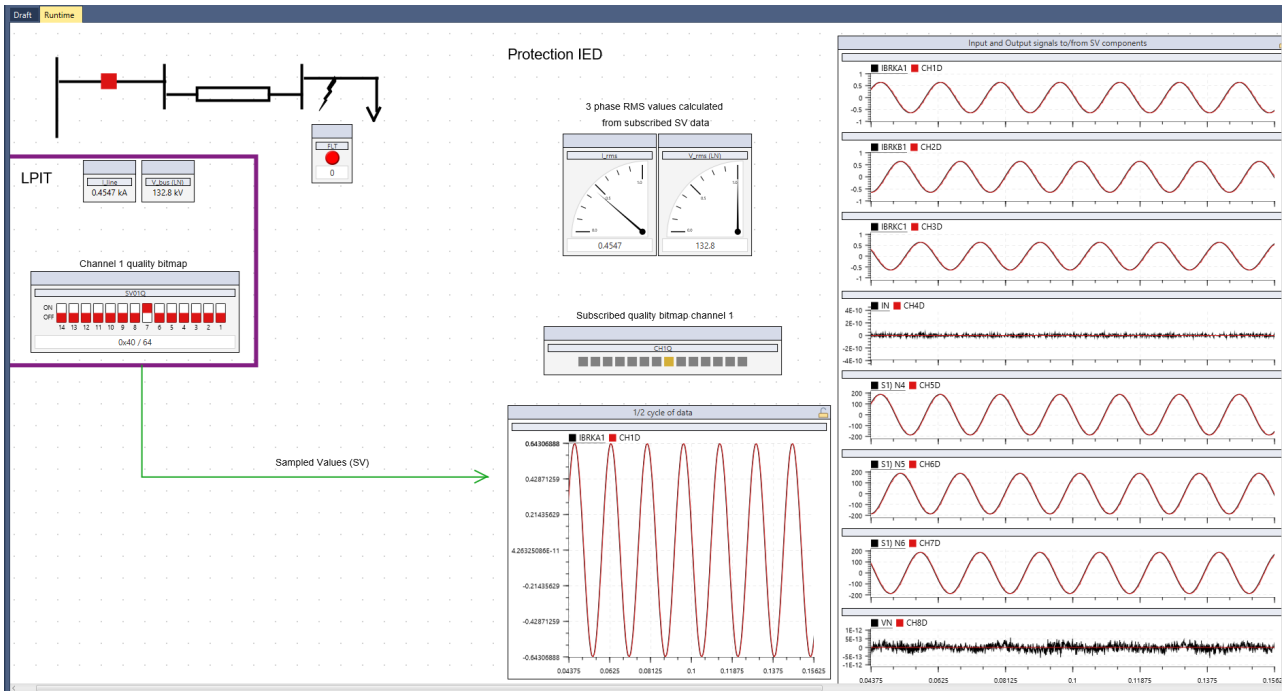
10.1. Example scenario: SV_Communication



The above case can be found in the Tutorials tab in RSCAD FX 2.0 at the following location:



The runtime without the attack should look similar to the image below:



Stating the new values to be assigned for Sampled Values

The modifications to the Sampled Values data are included in a file "ns3.conf". This file

should be located in `/etc/ns3` in the Linux machine. The format of the text are given below

```
protocol sample_value
<svID> <ASDU number> <DataIndex> <value> <quality>
```

Here the file parser uses the string "protocol sample_value" to identify that the data following that line are for SV frame modification. The next lines contain information on what to modify.

svID - The Sampled Value ID

ASDU number - The ASDU index in the frame if there are multiple of them

DataIndex - The index of the data within the ASDU

Value - Is the new parameter value to be assigned.

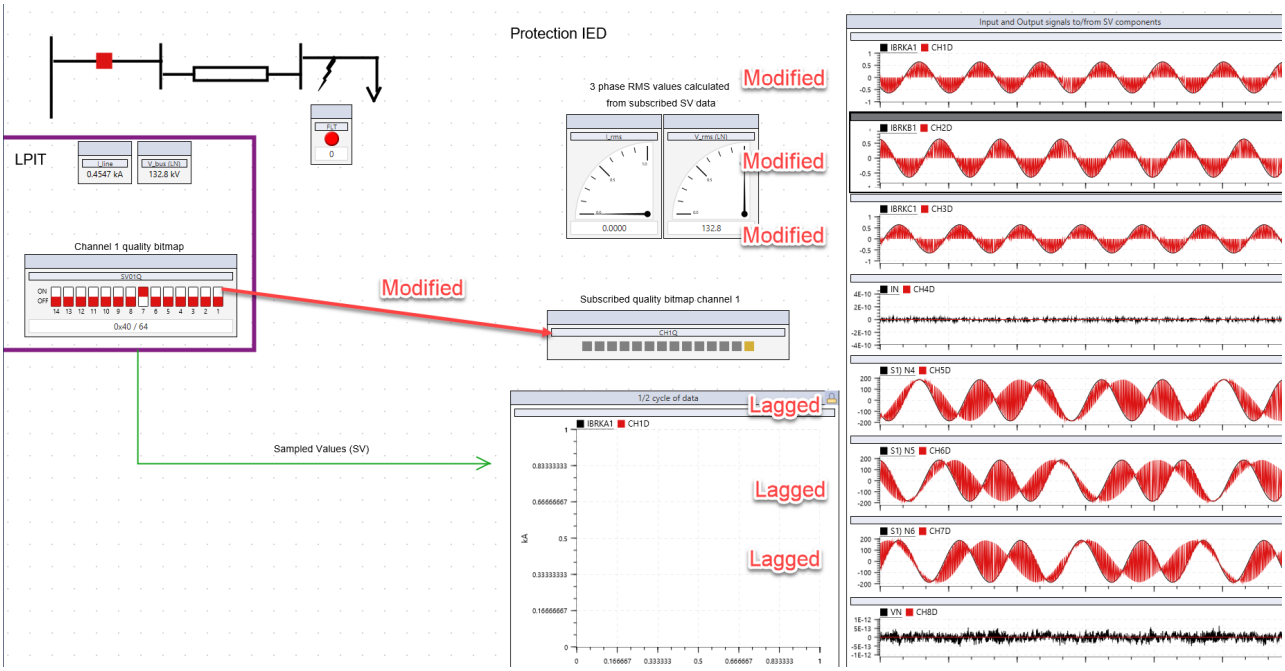
Quality - The new quality bit value

An example is shown below:

```
protocol sample_value
4001 0 0 0 1  \
4001 0 1 0  0\
4001 0 2 0 0
```

Here we are interested in the SV frames with the svID, `4001` and the ASDU index `0.` These data frames are created by the case that we are running. Each of these data frames that we create in the GTNET card for the case mentioned above, will have 8 data fields. In the first line we modify the first value to 0 and the quality to 1. In the next two lines we modify the second and third value to 0 and the quality to zero.

Now lets see this modifications in action. Run the simulation file `rtds-Tap-ICS-Mod-One_Net` in NS-3 using the instructions provided in section 5.1. Then start the RSCAD simulation. After running the simulation, the following can be observed in the runtime. If you stop the NS-3 simulation at any point you will see that the actual value that matches the values at the LPIT is shown at the Protection IED. As you can see, although we only modify the first three data values the current values are also seem changed that is because of the time lag for processing and transmission at NS-3 nodes. So an old value gets sent sometime later. Furthermore, the modification does not happen to all the SV frames. This is also because of the processing and transmission delay. For example the time difference between the frames shown below are 4ms.



Now lets study the wireshark capture taken at any interface connected to the network the GTNET card is connected. One observation you can make is that there is a duplicate frame created for each Ethernet multicast frame the Breaker IED sends. Now lets compare those Ethernet frames.

Original

```
Frame 980: 120 bytes on wire (960 bits), 120 bytes captured (960 bits) on interface enp7s0, id 0
Ethernet II, Src: RTDSTech_0b:5d (00:50:c2:4f:9b:5d), Dst: Iec-Tc57_04:01:a1 (01:0c:cd:04:01:a1)
802.1Q Virtual LAN, PRI: 4, DEI: 0, ID: 0
IEC61850 Sampled Values
    APPID: 0x4001
    Length: 102
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  savPdu
      noASDU: 1
    seqASDU: 1 item
      ASDU
          svID: 4001
          smpCnt: 1
          confRef: 1
          smpSynch: local (1)
        PhsMeas1
            value: 41
          quality: 0x00000000, validity: good, source: process
            value: -168
          quality: 0x00000000, validity: good, source: process
            value: 127
          quality: 0x00000000, validity: good, source: process
            value: 0
          quality: 0x00000000, validity: good, source: process
            value: 2460
          quality: 0x00000000, validity: good, source: process
            value: -9624
          quality: 0x00000000, validity: good, source: process
            value: 7164
          quality: 0x00000000, validity: good, source: process
            value: 0
          quality: 0x00000000, validity: good, source: process
```

Duplicate

```
Frame 1001: 120 bytes on wire (960 bits), 120 bytes captured (960 bits) on interface enp7s0, id 0
Ethernet II, Src: RTDSTech_0b:5d (00:50:c2:4f:9b:5d), Dst: Iec-Tc57_04:01:a1 (01:0c:cd:04:01:a1)
802.1Q Virtual LAN, PRI: 4, DEI: 0, ID: 0
IEC61850 Sampled Values
    APPID: 0x4001
    Length: 102
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  savPdu
      noASDU: 1
    seqASDU: 1 item
      ASDU
          svID: 4001
          smpCnt: 11
          confRef: 1
          smpSynch: local (1)
        PhsMeas1
            value: 0
          quality: 0x00000001, validity: invalid, source: process
            value: 0
          quality: 0x00000000, validity: good, source: process
            value: 0
          quality: 0x00000000, validity: good, source: process
            value: 0
          quality: 0x00000000, validity: good, source: process
            value: 2460
          quality: 0x00000000, validity: good, source: process
            value: -9624
          quality: 0x00000000, validity: good, source: process
            value: 7164
          quality: 0x00000000, validity: good, source: process
            value: 0
          quality: 0x00000000, validity: good, source: process
```

In additional to the changes in the data fields, you will see that the smpCnt field of the duplicate message is higher than that of the Original. This higher smpCnt ensures that this false SV frame gets accepted by the subscriber. However, it might also cause synchronization problems at the subscriber and it might get non-responsive.

11 CHANGES MADE TO NS-3.29

NS-3.29 is an open source network simulator. Therefore, it does not have built in support to run cyber attacks. However, it has all the necessary infrastructure to carryout some network level cyber attacks such as DoS attacks and MITM attacks. In this section, we discuss the additions we included in NS-3.29.

11.1. Writing an Application to Create a TCP server and Client

Although there are examples in the NS-3.29 on UDP client and server, there is no example on the TCP client and server communication. Here, we describe how an application involving TCP Client and server can be developed. A general guide on how to create new applications can be found in https://www.nsnam.org/wiki/HOWTO_make_and_use_a_new_application. In our current application, we are creating an application called MyApp which can accept, process and send TCP packets. The source code can be found in /examples/rtds-dos-simulation_TCP_BiDir.cc. The three way hand shake involved in a client sending a packet is handled automatically by the TcpSocketFactory. The only part we have to worry about is the acceptance of a packet in the server side. Every application has to implement three functions by default.

1. The Setup function
2. The StartApplication function
3. The StopApplication function

In the Setup function, we initialize the class variables. In the StartApplication function we run all the functions that are needed for the application to do its job. These functions continue to run until the function StopApplication is called. These two functions are called by the event scheduler when we set the start time and the stop time for the application at the caller. In addition to these functions, we have the following functions for the TCP application to process the packet data.

1. HandleAcceptRequest function
2. HandleAccept function

3. HandlePeerClose function
4. HandlePeerError function
5. HandleClose function
6. PrintTraffic function
7. pktProcessingIngressNode
8. pktProcessingEgressNode
9. pktProcessingAggregatorNode
10. giveParsingString

11.1.1. HandleAcceptRequest

This function is called when the TCP server gets a SYN request. Usually we do not need to do anything since the TCP protocol handles it.

11.1.2. HandleAccept

This function is called after the three way hand shake is done and we need to call the function to process the packet inside this function using callbacks. Furthermore, we can make the server drop the connection if the number of sessions are above a certain threshold.

11.1.3. HandlePeerClose

This function is called when the client send the FIN packet to close the socket connection.

11.1.4. HandlePeerError

This function is called when the client is in an error situation. We close the socket at the server at this situation.

11.1.5. HandleClose

This function is called when the server closes the connection.

11.1.6. PrintTraffic

This is one of the functions that processes the TCP data. This function is used to forward the data as it is

the simulated Aggregator node in the NS-3 simulation or the actual DER IP at the RTDS simulator.

11.1.7. pktProcessingIngressNode

This function processes TCP data at the ingress node. The main requirement of this function is to figure out the simulated node to forward the data. This is achieved by processing the first three integers in the packet data. The first one is the message type, the send is the index number of the DER or the aggregator the message was initialized, and the third is the index number of the DER or the Aggregator the message is destined to. If the Message type is eight, the message carries the flexibility information. Then the second integer is the ID of the DER and the last integer is the ID of the Aggregator. If the message type is 7, the message carries the setPoints. The second integer is the Aggregator index and the third is the DER index the setPoints are destined to.

11.1.8. pktProcessingEgressNode

This function forward the data to the correct GTNET IP the packets are destined. If the message type is 7, these messages are sent to the correct DER GTNET IP based on the third integer entry in the packet. If the message type is 8, it is forwarded to the correct Aggregator GTNET IP based on the third integer in the packet.

11.1.9. pktProcessingAggregatorNode

This function forwards the setPoints to the correct simulated DER based on the second integer in the packet.

11.1.10. giveParsingString

This function returns a string array of the information presented in each 4 byte block of the data area of the SKT application based on the message type.

11.2. Writing an Application to Create a TCP SYN flood attack

The source code for this SYN flood attack can be found in /src/applications/model/tcp-syn-flood.cc. Here we provide a brief explanation on how the code works. As explained earlier, in SYN flood attacks the client send TCP session initialization messages using fabricated source IPs. As any NS-3 application, this application has the tree functions that initiate, start and stop the application. The respective functions are: Setup, StartApplication and StopApplication. In the Setup function, we initialize the NS-3 node the application is installed, the IP of the victim, the actual IP of the node, the TCP port that is exploited and the

time between two consecutive SYN packets. In the StartApplication function, we create the socket and schedule an event to call the SendSyn function. The SendSyn function is the function that uses the socket to send the SYN packets. Then we have the StopApplication function which deallocates the socket and clear the data.

In the SendSyn function, we generate a bogus source IP and create the TCP header and the IPv4 header and send the packet to the IP address of the victim.

11.3. Writing an Application to change TCP and UDP packets on the Wire

These modifications enabled MITM type of attacks. The source code for this application can be found in /src/applications/model/attack-app.cc. In the Setup function, we initialize the NS-3 node the application runs in, the network device, the IPv4 interface, the actual IP address of the node interface, the victims IP address and the victims MAC address. In the StartApplication function, we use the device receive callback function to get the packets on the wire. We set the callback to the function NonPromiscReceiveFromDevice. Then when ever a packet is received in that device this function gets called. In this stop application function, we stop this application.

Inside the NonPromiscReceiveFromDevice function we call the ReceiveFromDevice function. Here, we remove the headers and modify the data such that only DER1 has flexibility.

11.4. Modifications made in arp-l3-protocol to enable ARP spoofing

The source code that implement the ARP protocol is in /src/internet/model/arp-l3-protocol. This code in ns-3.29 does not accept unsolicited ARP replies. Therefore we modified the code to accept a flag m_spoofARP which enables the device to reply to any ARP request with its own MAC address.

12 References

[1] Elmrabet, Z., Ghazi, H.E., Kaabouch, N., & Ghazi, H.E. (2018). Cyber-Security in Smart Grid: Survey and Challenges. Computers & Electrical Engineering, 67, 469-482.

[2] M. Korman, E. Mathias, O. Gehrke, A. M. Kosek . D1.1 Smart grid scenarios, version 1.1. Retrieved from Cyber-physical security for low-voltage grids website: http://www.salvage-project.com/uploads/4/9/5/5/49558369/salvage_d1.1_v1.1.pdf

[3] R. Liu, C. Vellaithurai, S. S. Biswas, T. T. Gamage and A. K. Srivastava, "Analyzing the Cyber-Physical Impact of Cyber Events on the Power Grid," in IEEE Transactions on Smart Grid, vol. 6, no. 5, pp. 2444-2453,Sept.2015. doi: 10.1109/TSG.2015.2432013

[4] B. Chen, K. L. Butler-Purry, A. Goulart and D. Kundur, "Implementing a real-time cyber-physical system test bed in RTDS and OPNET," 2014 North American Power Symposium (NAPS), Pullman, WA, 2014, pp. 1-6. doi: 10.1109/NAPS.2014.6965381

[5] A. Hahn, A. Ashok, S. Sridhar and M. Govindarasu, "Cyber-Physical Security Testbeds: Architecture, Application, and Evaluation for Smart Grid," in IEEE Transactions on Smart Grid, vol. 4, no. 2, pp. 847-855, June 2013. doi: 10.1109/TSG.2012.2226919