NetSim Interfacing with Node Red and IBM Watson

Project Download Link:

https://github.com/NetSim-TETCOS/Dynamic_Traffic_Light_Control_in_NetSim_VANETs_v13.0/archive/refs/heads/main.zip

Objective:

Interfacing NetSim with IBM node red and IBM Watson IoT cloud platform.

About Node-Red and IBM- Watson QuickStart

a. **Node-RED** is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.

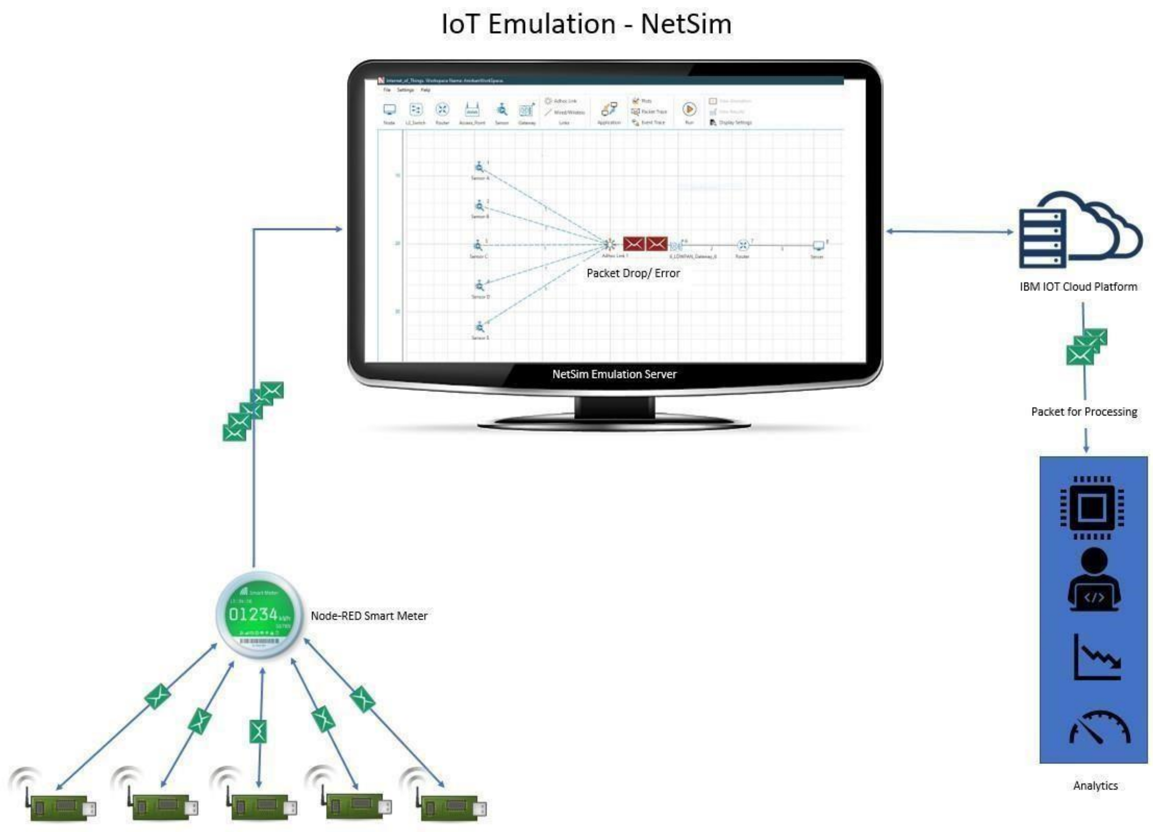
   Node-RED is built on Node.js, taking full advantage of its event-driven, nonblocking model. This makes it ideal to run at the edge of the network on lowcost hardware such as the Raspberry Pi as well as in the cloud

b. **Quickstart** is a way of getting a single device sending data to Watson IoT Platform that you can visualize in your browser. No sign up is required, you can see how easy it is to connect your device and, if you want to continue after Quickstart, you can sign up, add security to your device and you are ready to go.

System and Hardware Requirements

1. NetSim Standard version with Emulator (v12.2)
2. Node Red Installer - It is a flow editor which provides virtual sensor environment
3. Two Systems
   a. System 1: Sensor Server & NetSim Emulation Server Installed
   b. System 2: Sensor Client Installed

High Level Connectivity Diagram



- IBM node red 'virtual sensors' are mapped to sensors in NetSim.
- The traffic sent by the IBM node-red virtual sensosr is routed inside NetSim and sent from the sensors to the sink in NetSim's virtual network.
- From the sink it is sent to IBM Watson IoT platform cloud server.

Step by step implementation.

System 1 Setup: Sensor server and NetSim Emulation Server Installation

Node-Red Installation and its working

- Installation of Node Red on Windows: Follow the steps on how to install Nodered in windows, at https://nodered.org/docs/getting-started/windows
- Running Node-Red and configuring Sensor server:
- Run Node-red through cmd window
- Open browser http://127.0.0.1:1880/?#flow/
- Install node-red-contrib-scx-ibmiotapp and node-red-dashboard package

Setting up Sensor - Server

- Design the flow graph as shown
    i. Fig:1.1: If you want to have sensor server
    ii. Fig:1.2: If you are using Smart meter to get the payload to sensor server
- Configure the device present in the flow graph in Figure 1.1, Figure 1.2, Figure 1.3

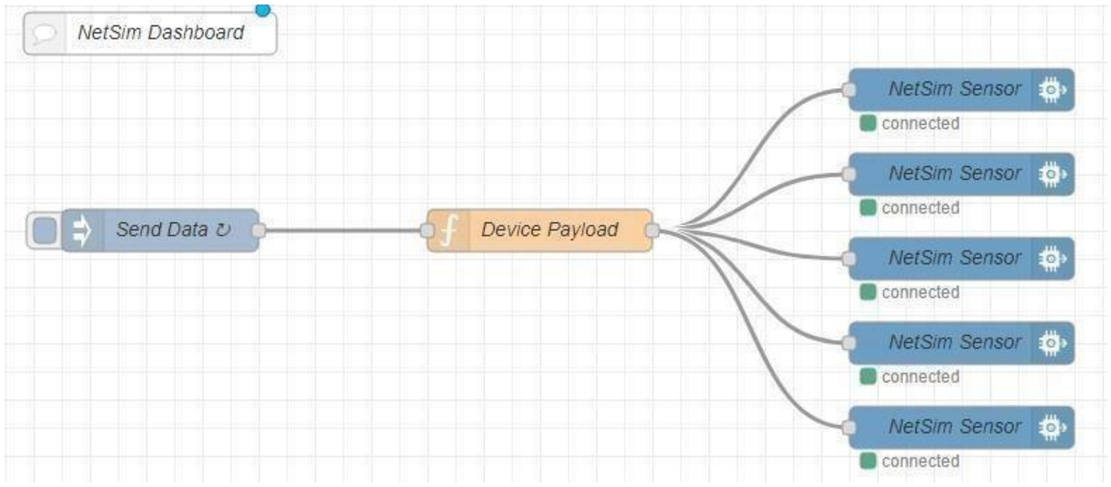


Figure 1.1 Flow design for Multiple sensors

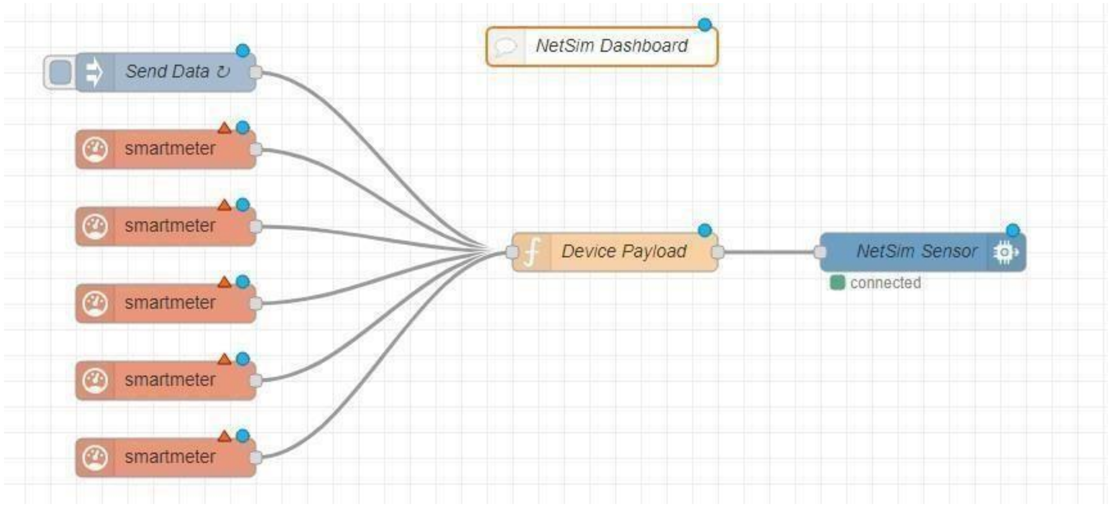
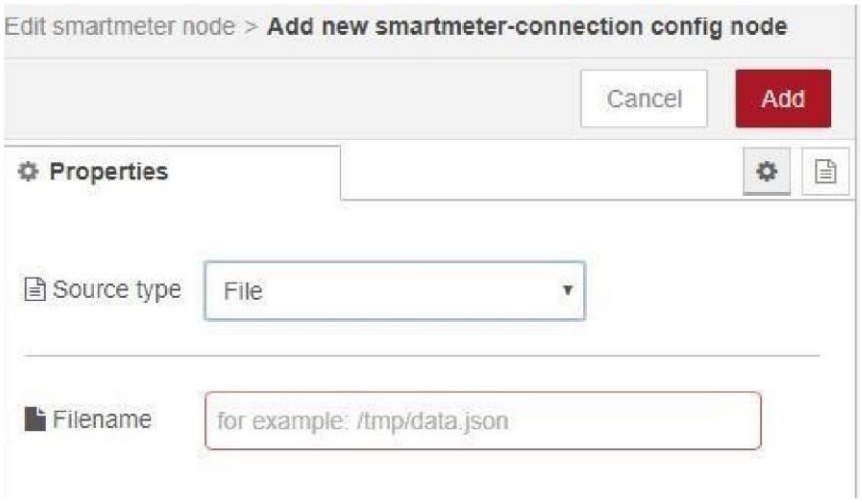


Figure 1.2 Multiple Smart Meter connecting one Virtual sensor

Users can also use a smart meter to send the data. Which accepts the following as input and here is a format to give file as input as shown in Figure 1.3

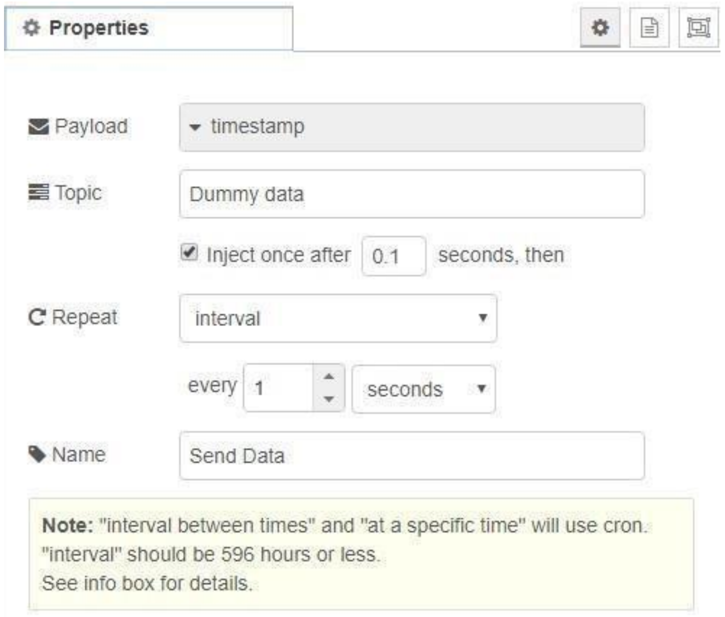
a. HTTP
b. USB
c. FILE


Edit smartmeter node > **Add new smartmeter-connection config node**

Source type: File

Filename: for example: /tmp/data.json

Figure 1.3



Payload: timestamp

Topic: Dummy data

☑ Inject once after 0.1 seconds, then

Repeat: interval

every 1 seconds

Name: Send Data

Note: "interval between times" and "at a specific time" will use cron.
"interval" should be 596 hours or less.
See info box for details.

Figure1.4 Sensor Data: Generating 1Packet/second which accepts decimal values

- Since we are now configuring a virtual sensor, the device payload is set by writing a function in Device Payload in flow chart as Figure 1.5 Sensor Payload

  Function:

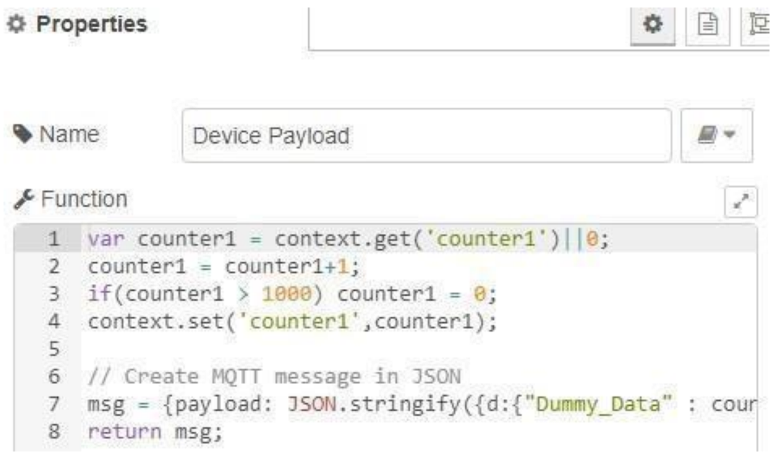
  ```
  var counter1 = context.get('counter1')||0;
  ```

counter1 = counter1+1; if(counter1
> 1000) counter1 = 0; context.set('counter1',counter1);

// Create MQTT message in JSON msg = {payload:
JSON.stringify({d:{"Dummy_Data" : counter1}})}; return
msg;

When having multiple sensors in the scenario, users can generate Payload by specifying port number for each sensors.

Users need to add ***msg.port=<port number>;*** *in the* payload function



```
var counter1 = context.get('counter1')||0;
counter1 = counter1+1;
if(counter1 > 1000) counter1 = 0;
context.set('counter1',counter1);

// Create MQTT message in JSON
msg = {payload: JSON.stringify({d:{"Dummy_Data" : cour
return msg;
```

Figure 1.5 Sensor Payload

IOT Emulation using virtual Smart-meter using IBM Node-red

Users can also use Node-red Smart-meter to generate payload instead of writing virtual sensor function.
Node-red requires additional installation of packages related to smart-meter.

Node-RED Node, that reads and parses the data from smartmeter devices. Supports for example Hager eHz Energy Meter, EMH Energy Meter, EFR SmartGridHub, Siemens 2WR5, Elster AS1440, Iskraemeco MT174, Itron EM214 Typ 720 etc.

For additional information on how to install smart-meter and use that as payload generator look into https://flows.nodered.org/node/node-red-contrib-smartmeter.

Sending data from real sensor

Users can also use real sensor instead of virtual sensors. i.e. users can connect real sensors to Raspberry Pi and then install Node- red on Raspberry Pi and configure the Node-flow.

While adding function payload users can directly retrieve the real sensor values and send them to cloud.

Here is an example on how users can connect real sensors to IBM Watson IoT Platform.

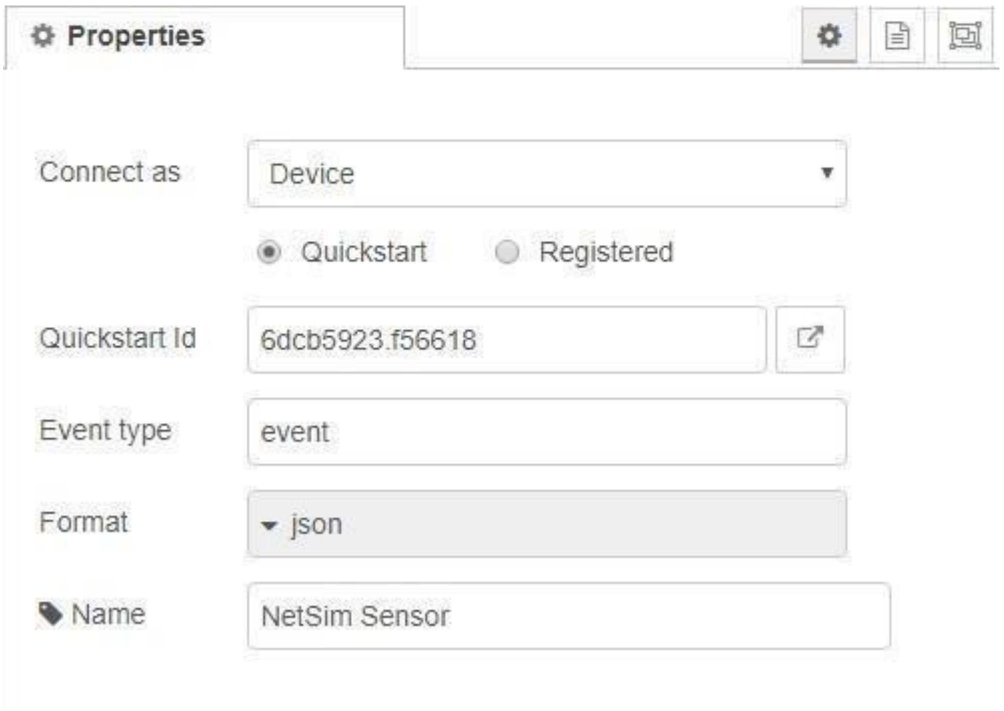
https://www.hackster.io/madoyon/connect-sensors-to-ibm-watson-iot-platform-083973

Encryption and Decryption of sensor data

The packet payload can also be encrypted if required. No encryption is applied to the payload by default. Node-RED nodes uses CryptoJS to encrypt and decrypt messages.

For detail on how to encrypt and decrypt the message users can refer to the examples at https://flows.nodered.org/node/node-red-contrib-crypto-js

- Connect NetSim Senor node to IBM Watson QuickStart IoT Platform



Connect as	Device
	[✓ Quickstart] Registered
Quickstart Id	6dcb5923.f56618
Event type	event
Format	json
Name	NetSim Sensor

Figure 1.6

Fig:1.6

- Deploy the Node-Red Flow
- Open the Link icon present in Figure 1.6 to view live sensor data the payload that you set.

Installation of NetSim Emulator and setting up emulation server.

- Install NetSim as per NetSim Installation Guide and Start NetSim as administrator (Make sure that Emulator licenses are available)
- Open IoT network module and Create a simple scenario with one sensor connecting to cloud as shown in Figure 1.7
- Create Emulation application between Sensor and Cloud (Also users can specify port if known) as shown in Figure 1.8

i. Source Real IP: 192.168.0.20 (System running Node Red) ii. Destination Real IP:0.0.0.0 (Mapping traffic to IBM Watson cloud)

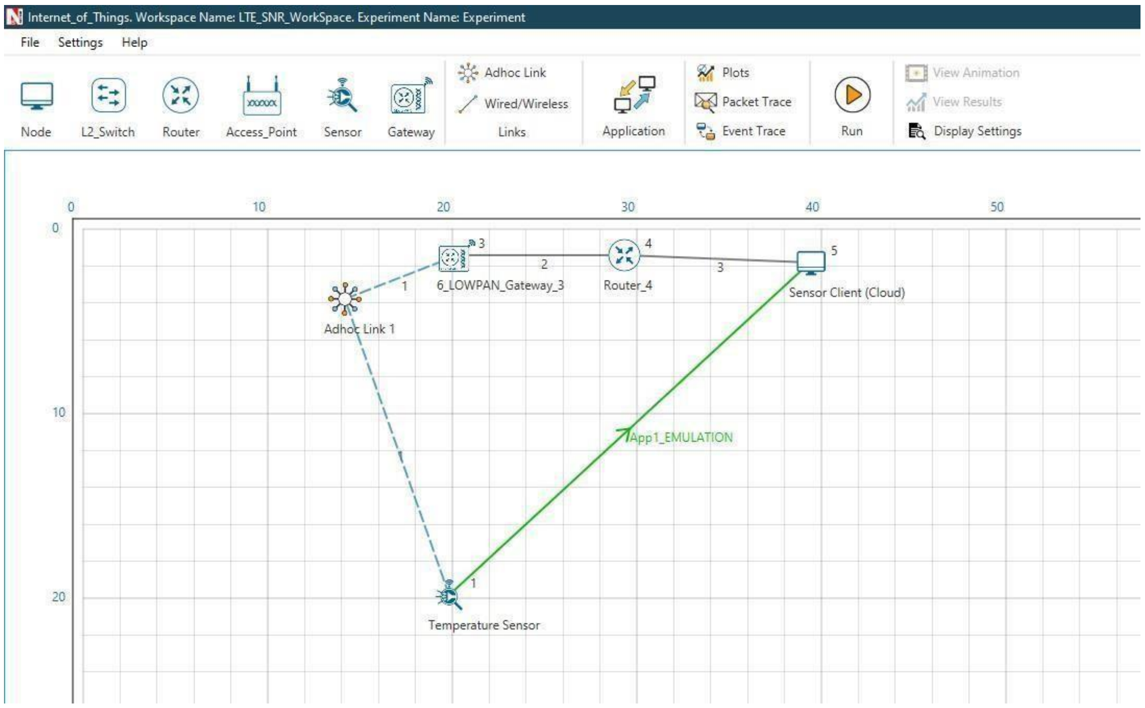
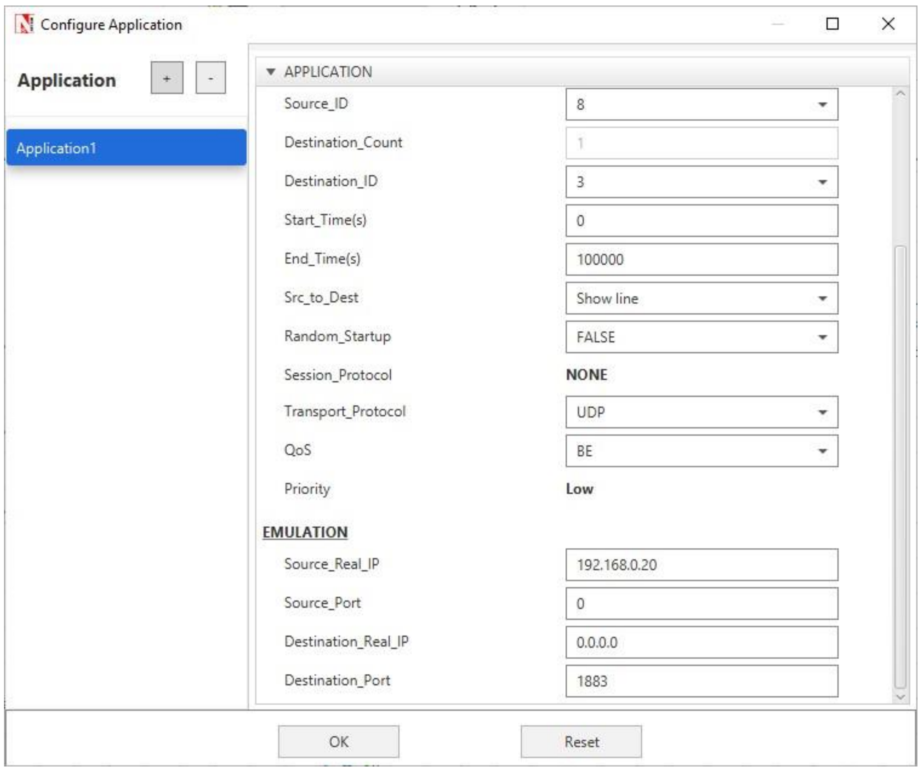


Figure 1.7: Network designed in NetSim

- Also place the sensor and the gateway at suitable distances for pathloss to have an impact on simulation performed.
- Now Run the Simulation for 100s



Source_ID	8
Destination_Count	1
Destination_ID	3
Start_Time(s)	0
End_Time(s)	100000
Src_to_Dest	Show line
Random_Startup	FALSE
Session_Protocol	NONE
Transport_Protocol	UDP
QoS	BE
Priority	Low

EMULATION

Source_Real_IP	192.168.0.20
Source_Port	0
Destination_Real_IP	0.0.0.0
Destination_Port	1883

Figure 1.8: NetSim Application Panel

System 2 Setup: Sensor Client

a. Install Node-red and perform its running as it was explained above for Sensor client
b. Design the flow graph as shown in Figure 1.9
c. Configure the device present in the flow graph in Figure 1.10, Figure1.11 and Figure 1.12

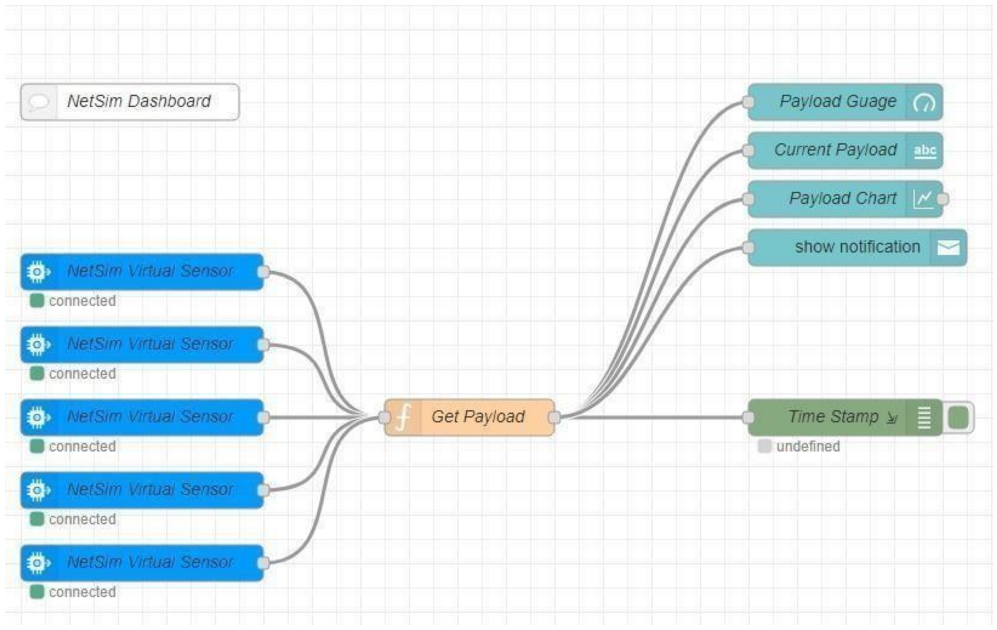
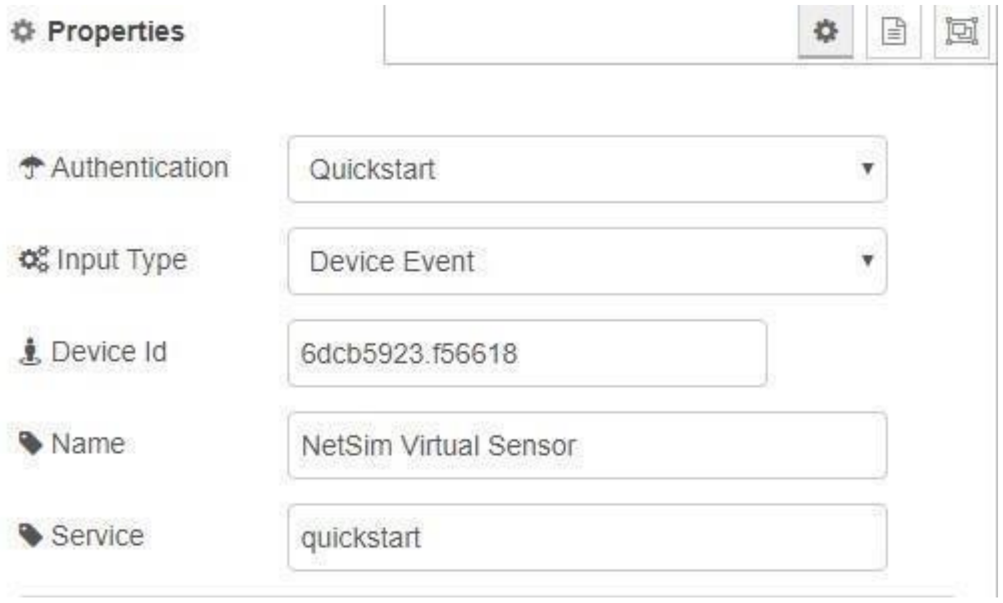


Figure 1.9

d. Configure NetSim Virtual Senor Node as shown in Fig:1.9



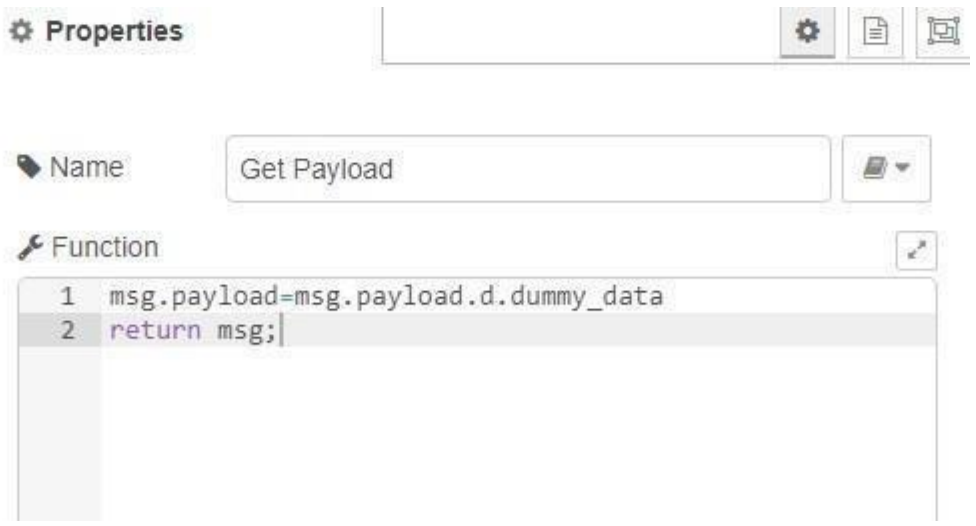
Properties

Authentication	Quickstart
Input Type	Device Event
Device Id	6dcb5923.f56618
Name	NetSim Virtual Sensor
Service	quickstart

Figure 1.10

e. To access the payload that we generated in Sensor cloud use the Get payload Function as shown in Figure1.11:

```
msg.payload=msg.payload.d.dummy_data
        return msg;
```

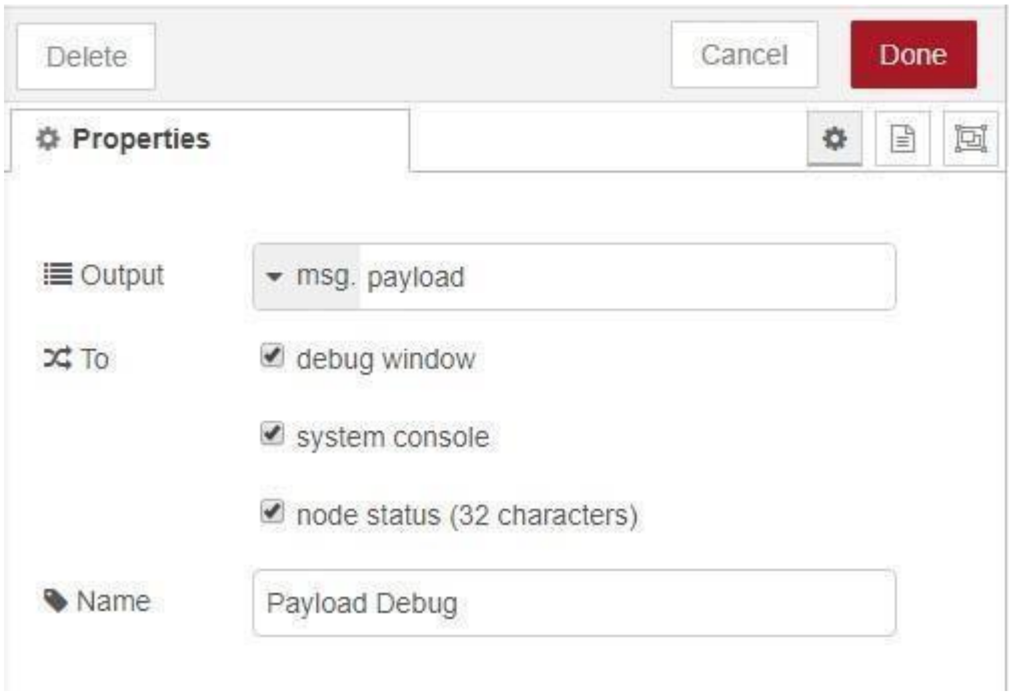


Get Payload

```
msg.payload=msg.payload.d.dummy_data
return msg;
```

Figure1.11

f. Enable Debug window using Debug node to access payload to file as shown in Figure 1.12. If you need that payload to be written to a file, then while running Node-red command to the beginning you can start writing it to a file as
node-red >getpayload.txt



Output: msg. payload

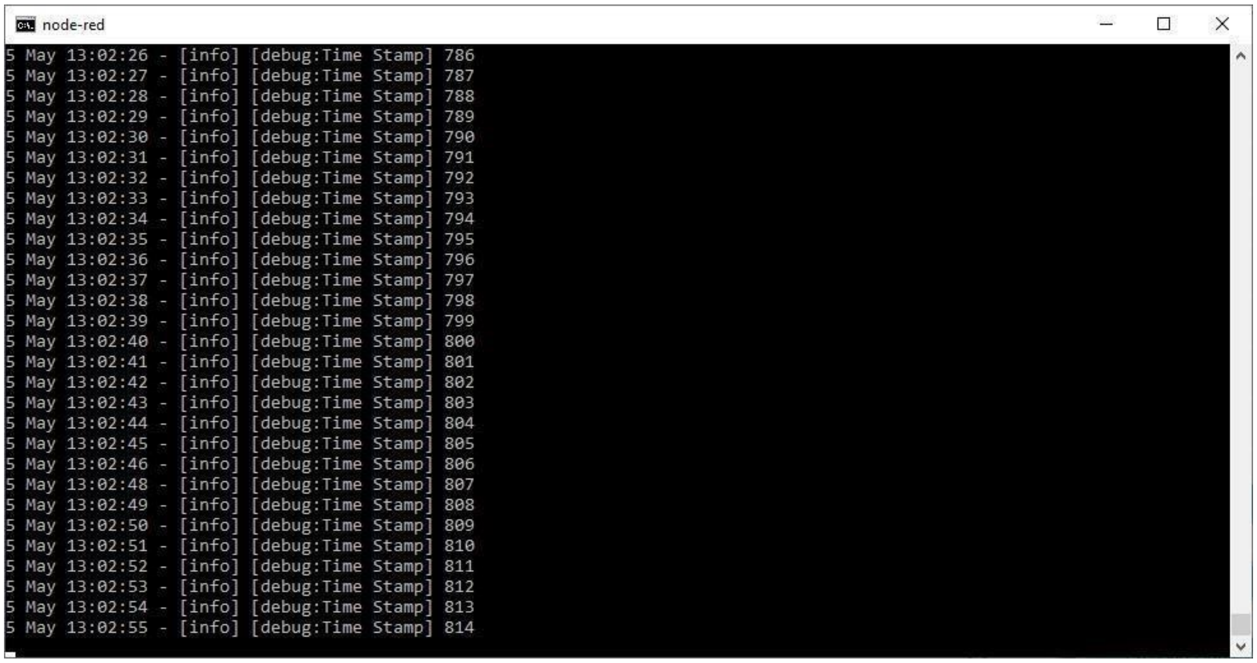
To: [✓ debug window] [✓ system console] [✓ node status (32 characters)]

Name: Payload Debug

Figure 1.12

g. Deploy the Node-Red Flow

h. Make sure that you start receiving the data through the console as shown in Figure 1.13


node-red

```
5 May 13:02:26 - [info] [debug:Time Stamp] 786
5 May 13:02:27 - [info] [debug:Time Stamp] 787
5 May 13:02:28 - [info] [debug:Time Stamp] 788
5 May 13:02:29 - [info] [debug:Time Stamp] 789
5 May 13:02:30 - [info] [debug:Time Stamp] 790
5 May 13:02:31 - [info] [debug:Time Stamp] 791
5 May 13:02:32 - [info] [debug:Time Stamp] 792
5 May 13:02:33 - [info] [debug:Time Stamp] 793
5 May 13:02:34 - [info] [debug:Time Stamp] 794
5 May 13:02:35 - [info] [debug:Time Stamp] 795
5 May 13:02:36 - [info] [debug:Time Stamp] 796
5 May 13:02:37 - [info] [debug:Time Stamp] 797
5 May 13:02:38 - [info] [debug:Time Stamp] 798
5 May 13:02:39 - [info] [debug:Time Stamp] 799
5 May 13:02:40 - [info] [debug:Time Stamp] 800
5 May 13:02:41 - [info] [debug:Time Stamp] 801
5 May 13:02:42 - [info] [debug:Time Stamp] 802
5 May 13:02:43 - [info] [debug:Time Stamp] 803
5 May 13:02:44 - [info] [debug:Time Stamp] 804
5 May 13:02:45 - [info] [debug:Time Stamp] 805
5 May 13:02:46 - [info] [debug:Time Stamp] 806
5 May 13:02:48 - [info] [debug:Time Stamp] 807
5 May 13:02:49 - [info] [debug:Time Stamp] 808
5 May 13:02:50 - [info] [debug:Time Stamp] 809
5 May 13:02:51 - [info] [debug:Time Stamp] 810
5 May 13:02:52 - [info] [debug:Time Stamp] 811
5 May 13:02:53 - [info] [debug:Time Stamp] 812
5 May 13:02:54 - [info] [debug:Time Stamp] 813
5 May 13:02:55 - [info] [debug:Time Stamp] 814
```

Figure 1.13

Routing Traffic Through NetSim (Emulation)

Once both the systems are setup and the client starts receiving sensor data that was set in the server, we are ready to perform NetSim Emulation.

Run NetSim Emulation for all the cases mentioned below and create a log of the debug window when receiving the sensor data.

Example: Impact of variation of channel conditions and sensor count on packet delivery ratio

Note: Since we are performing Emulation the results will vary count of received packets.

Case 1: Sensor count 1

Channel Characteristics	Emulation Application 1 Packet received out of 100
No Pathloss	89
Pathloss - 2	70

Pathloss - 3	58
Pathloss - 3.5	28

Case 2: Sensor count 2

Channel Characteristics	Emulation Application 1	Emulation Application 2
No Pathloss	75	80
Pathloss - 2	73	78
Pathloss - 3	77	63
Pathloss - 3.5	32	27

Case 3: Sensor count 5

Channel Characteristics	App 1	App 2	App 3	App 4	App 5
No Pathloss	63	59	68	44	61
Pathloss - 2	55	50	52	41	51
Pathloss - 3	31	35	28	26	49
Pathloss - 3.5	19	8	12	6	13

Case 4: Sensor count 10

Channel Characteristics	1	2	3	4	5	6	7	8	9	10
No Pathloss	35	32	39	39	46	43	38	34	45	48
Pathloss - 2	30	28	31	34	39	41	33	20	41	18
Pathloss - 3	9	5	10	8	3	5	2	15	11	5
Pathloss - 3.5	5	0	6	2	0	0	0	8	5	0