An SDN Solution For a GEO and LEO Satellite Transceiver Based on GNU Radio Companion

Ahmed Bannour, Veronique Moeyaert UMONS, Faculty of Engineering
Electromagnetism and Telecommunications Department, MONS, Belgium
Email:{ahmed.bannour,veronique.moeyaert}@umons.ac.be

***Abstract*—Nowadays a novel satellite communication solution has to be introduced for the coming satellite networks. Software Defined Networking (SDN) and Software Defined Radio (SDR), can play a key role to innovate and improve satellites-terrestrial communications. GNU radio as SDR and Mininet as SDN, offer the ability to manage the network satellite resources. Through scenario and use cases, this paper provides an in-depth analysis for an effective combination of SDN, GEO, and LEO satellites using GNU Radio simulation and Python3 scripting. This new architecture can bring satellite communications towards a new proposal of data transmission technology to delimit the various potential areas of improvement. That can be pursued by the added value of SDN-GNU Radio technologies in the field of the satellite ground segment. To achieve this result, the newly introduced architecture can be fully controlled at the physical layer level. The solution is validated by the deployment of three scenarios: with SDN, without SDN, and SDN adding a streaming overload. Based on the results of the presented simulation, we found that our architecture can achieve better quality of service through efficient satellite traffic control, when SDN and GNU Radio are deployed.**

***Index Terms*—Integration of Satellite and Terrestrial Networks, Next Generation Networks, Software Defined Networking, Space Networking, GNU Radio.**

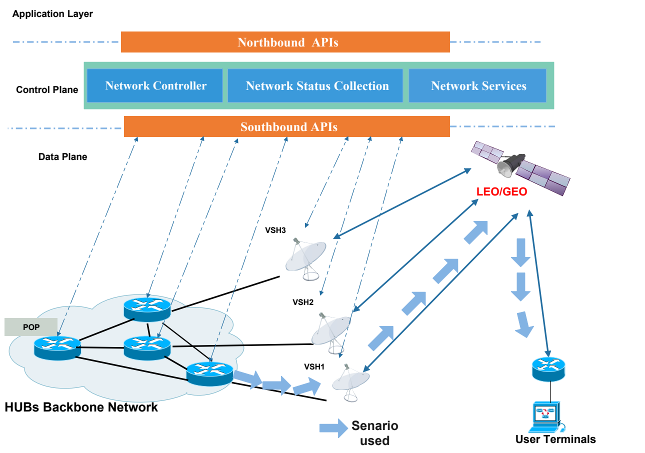


Fig. 1: SDN-GNU Radio experiment scenario for GEO/LEO satellites

I. Introduction

Radio-link availability and services reliability are now highly demanded for the future satellite networks [1], [2].

For example, Low Earth Orbit (LEO) are widely used for Earth observation. The rapid movement of these satellites makes them visible in a given location for a very short period of time, about ten minutes. They must therefore be autonomous to fulfill their mission being rarely in contact with their control center. Another important example is the Geostationary Earth Orbit (GEO). It is by far the most widely used, especially with the development of telecommunications needs, (e.g., it is where live broadcast satellite are located).

Beside their benefits, GEO/LEO satellite configuration and monitoring are yet to be improved to ensure good connectivity and fast configuration. For example, access to configure GEO/LEO satellite is not available unless the satellite is detected by the terrestrial station. Again, the current satellite networking solution is not centralized. Hence robust synchronization and scalability is not very well ensured.

The concept of Software Defined Network (SDN) has dramatically changed the field of modern wireless telecommunications networks. This concept is based on new technologies stepping towards miniaturization and the increase of intelligence of tools and machines. The OSI layered model in the current wireless networks makes the control and the management of the infrastructure complex and hard to handle. This is due to the combination of the required data with data responsible for the control. SDN creates an opportunity to solve this problem. Indeed, the global network vision is logically centralized in the control plane and the packet delivery is very efficient in the data plane. Software Defined Network brings flexibility, automation and network customization to networks. Therefore, this type of network also represents an opportunity to facilitate the deployment and management of different wireless networks, things that the previous OSI model cannot offer. SDN has successfully paved the way for next-generation networking, but research work based on wireless networking combined with SDN has recently started.

Henceforth, the combination of Software Defined Network (SDN) and GNU Radio can give a robust and manageable satellite network solution. This can introduce new directives to solve satellite network drawbacks [3]–[5]. The main idea in this paper is to enhance the satellite networking using the new concept of SDN. This new architecture will ensure the decoupling of data, responsible for control from transmission equipment using the controller Python network Operating X-system (POX).

In SDNs, the control plane is placed in a centralized controller that has visibility over the entire network, including the hosts connecting to it. Applications that tell controllers how to program the network using programming interfaces are called ”NorthBound” while the Application Programming Interfaces

(APIs) and protocols used by the controller to communicate with network equipment are called "SouthBound" (Fig. 1). The controller POX uses OpenFlow protocol, to administer switches and network equipments.

An OpenFlow network consists of at least one OpenFlow switch and a controller that programs it dynamically. OpenFlow defines a set of criteria for selection on packages required to handle networking operation using Python. These criteria can be non-exhaustive (i.e., the input interface used, the MAC address, the IP address or port number of a packet, both source and destination, or metadata associated with the packet). Next to the criteria for selecting a package, OpenFlow defines actions that can be performed on them. Among possible actions, one can cite:

- Send the packet to an exit gate, a set of gates (group) or to the controller.
- Define the queue in which the packet will be placed in the exit door.
- Modify the packet: header fields, tags, etc ...
- Delete the packet and end its processing.

Logic deterministic packet handling is contained in the switch flow tables. These tables are made up of rules that the controller adds (and removes) dynamically. Each rule includes a set of selection criteria, a priority, and instructions. When a packet arrives at the switch, it is initially assigned empty set of actions. The switch then cycles through the rules in its first table in descending order of priority, looking for a criterion that matches the packet. As soon as the packet is selected by a rule, the cycling is interrupted and the instructions of the rule are executed [6].

A defined satellite networks solution is presented in [7]–[9]. This new OpenSAN architecture gives a fluid control and high efficiency. Operationally Responsive Space (ORS) is introduced by Feng *et al* in [10]. Compared to the latter, this reference paper gives more insights when dealing with emergency response in satellite networking communications. ORS uses OpenFlow protocol in controller to ensure a centralized space to ground network technologies. However the proposed solution does not support the full network visualization.

The related research works, are limited to theoretical analysis, without giving a sufficient realistic SDN satellite networks model to support their concepts. To the best of our knowledge, the combination of GNU radio with SDN to support satellite communications is proposed for the first time in this paper. We can find SDN in some related work enabled in ground satellite portion [11]. However, those solutions degrade network monitoring and reduce flexibility. In this work, we deploy SDN in both ground and space satellite communications portions. This solution will ensure good traffic management and easy configuration. GEO and LEO satellites will be fully controllable in a centralized manner with a flexible global access.

The research work is inspired from the SDR Makerspace (https://sdrmaker.space/), which is an initiative of the European Space Agency and the Lieber Space Foundation that brings together researchers from around the world. In fact, the GEO and LEO project uses GNU radio unit that simulates earth-satellite system communications and a variety of aberrations that a satellite channel can bring. For example, the Carrier Frequency Offset (CFO) due to the Doppler effect (in 435 MHz band), the loss of the variable free space path due to the path of the satellite, or the atmospheric effects (such as atmospheric gases and precipitation) along the flight, have been shown to cause a significant deterioration in the communication quality.

To summarize, we have made the following contributions:

- We enable GEO and LEO satellite communication under SDN architecture using Python3.
- The proposed topology is totally configurable and controllable through POX controller using Openflow protocol.
- A cross layer solution is presented, because transmission is fully configurable from the bit order (physical layer) with GNU radio scripting to logical order (IP traffic) with SDN.

We sectioned the paper as follows. Section II presents the proposed architectures of satellite networks. Then, Section III is devoted to the analysis of each of the scenarios identified. Finally, the main conclusions are elaborated in Section IV

II. The Proposed Architecture

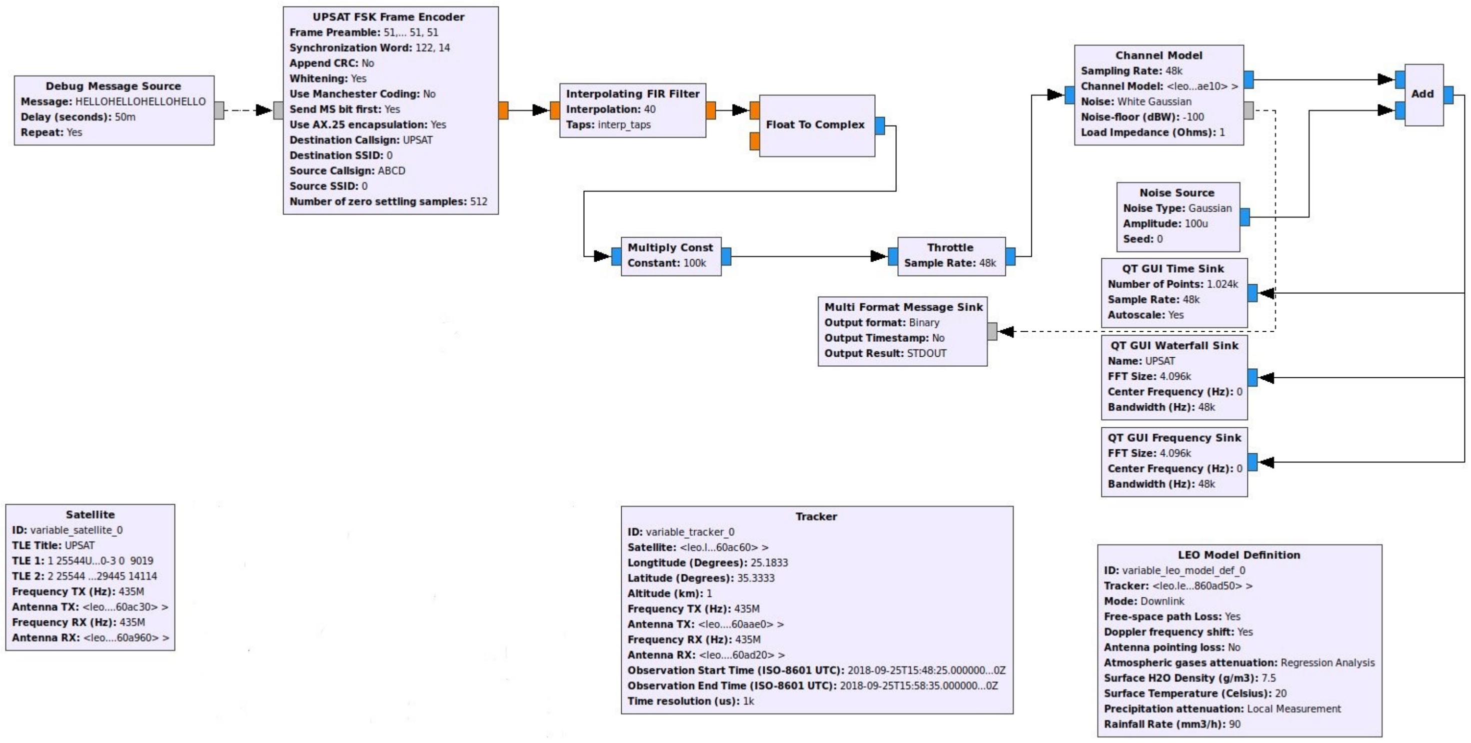
Fig. 1 illustrates an overview of the proposed topology. The architecture is sliced into three planes, the *Data Plane*, the *Control Plane*, and the *Application plane*.

The *Data Plane*. As it was seen in the traditional network principle, the processing resources on satellites should be limited. However, for the SDN network the controller is responsible for policies linked to routing, security, management, and resource allocation through a table flow. In other words, it is the maestro of the entire network. The POX controller then develops policies in relation with GEO and LEO satellites needs and user requirements. These policies are sent to the terminals satellite GEO/LEO to direct traffic between satellites.

The *Control Plane*. It sets up a SDN controller on the GEO/LEO orbit satellites, taking into account the general behavior of the entire topology with respect to network reliability. GEO/LEO satellites collect information about the status of links between satellites and send it to POX controller for the management. During this time, the instructions for network traffic are transmitted to the GEO/LEO satellites. An OpenFlow chart opens, when rules are enabled at satellites level and redirected again to end users.

The *Application Plane*. It presents the programs that define the logic control of the network. The programming interface responsible is called 'Northbound API'. In general the application plane is responsible for ensuring the Quality of Service.

We implement the architecture in a software prototype as shown by the dashed arrow in (blue), with the Southbound API in the Fig. 1. Two nodes are used and each node acts as a router in the topology. Each node must be linked with SDN architecture and the communication between the various physical computers (end user) will be guaranteed through POX controller. Henceforth, the different instances can establish communication between one another and transmit packets via SDN. As illustrated in Fig. 1 the architecture presents two



Debug Message Source
Message: HELLOHELLOHELLOHELLO
Delay (seconds): 50m
Repeat: Yes

UPSAT FSK Frame Encoder
Frame Preamble: 51,... 51, 51
Synchronization Word: 122, 14
Append CRC: No
Whitening: Yes
Use Manchester Coding: No
Send MS bit first: Yes
Use AX.25 encapsulation: Yes
Destination Callsign: UPSAT
Destination SSID: 0
Source Callsign: ABCD
Source SSID: 0
Number of zero settling samples: 512

Interpolating FIR Filter
Interpolation: 40
Taps: interp_taps

Float To Complex

Multiply Const
Constant: 100k

Throttle
Sample Rate: 48k

Channel Model
Sampling Rate: 48k
Channel Model: <leo...ae10> >
Noise: White Gaussian
Noise-floor (dBW): -100
Load Impedance (Ohms): 1

Add

Noise Source
Noise Type: Gaussian
Amplitude: 100u
Seed: 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 48k
Autoscale: Yes

Multi Format Message Sink
Output format: Binary
Output Timestamp: No
Output Result: STDOUT

QT GUI Waterfall Sink
Name: UPSAT
FFT Size: 4.096k
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

QT GUI Frequency Sink
FFT Size: 4.096k
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

Satellite
ID: variable_satellite_0
TLE Title: UPSAT
TLE 1: 1 25544U...0-3 0 9019
TLE 2: 2 25544 ...29445 14114
Frequency TX (Hz): 435M
Antenna TX: <leo....60ac30> >
Frequency RX (Hz): 435M
Antenna RX: <leo....60a960> >

Tracker
ID: variable_tracker_0
Satellite: <leo.l...60ac60> >
Longtitude (Degrees): 25.1833
Latitude (Degrees): 35.3333
Altitude (km): 1
Frequency TX (Hz): 435M
Antenna TX: <leo....60aae0> >
Frequency RX (Hz): 435M
Antenna RX: <leo....60ad20> >
Observation Start Time (ISO-8601 UTC): 2018-09-25T15:48:25.000000...0Z
Observation End Time (ISO-8601 UTC): 2018-09-25T15:58:35.000000...0Z
Time resolution (us): 1k

LEO Model Definition
ID: variable_leo_model_def_0
Tracker: <leo.le...860ad50> >
Mode: Downlink
Free-space path Loss: Yes
Doppler frequency shift: Yes
Antenna pointing loss: No
Atmospheric gases attenuation: Regression Analysis
Surface H2O Density (g/m3): 7.5
Surface Temperature (Celsius): 20
Precipitation attenuation: Local Measurement
Rainfall Rate (mm3/h): 90

Fig. 2: GEO/LEO satellites transceiver based on GNU Radio Companion

APIs: Southbound API, and Northbound API. Southbound API interface represents the communication interfaces, which allows the SDN controller to interact with the infrastructure layer, such as switches, computers and routers. To maintain good communication with the South interface, OpenFlow protocol was enabled. Northbound APIs are used to program the transmission equipment, using the network abstraction provided by the control plane.

The Work Flow

- We start by creating an SDN topology with a Python3 scripting. The first step is to install Mininet (http://mininet.org/) to create the proposed architecture. Mininet contains the SDN controller POX which is responsible for the whole management of the network. We have also implemented the POX controller that can support OpenFlow protocol and it is written in Python3.
- Then we implemented a Python3 script, which interacts directly with GNURadio Companion to ensure GEO/LEO satellite transmission chain in the SDN topology as illustrated in Fig. 2.
- Finally we evaluated the performance of the prototype compared to traditional satellite transmission with three scenarios: without SDN, with SDN only and finally SDN with streaming load.

The ground earth station is represented in Fig. 2 blocs, as an instance of the *Tracker class*. It contains parameters for all the information needed to fully describe a ground station that follows a satellite in orbit, such as:

- The coordinates of the ground station (Latitude, Longitude and Altitude)
- TX / RX operating frequencies
- TX / RX antenna objects
- The observation period in UTC ISO-8601 format
- The resolution of the observation time.

A satellite is represented in Fig. 2, as an instance of the class Satellite. It contains parameters for all the information required to describe a satellite in orbit, such as:

- An orbit description.
- Satellite operating TX / RX frequencies
- TX / RX antenna objects.

Different types of antennas are supported by Fig. 2, including *Yagi, Helix, Monopole, Dipole* and *Parabolic reflector*. Each is described by a $C++$ class which extends the parent generic antenna and appropriately implements pure virtual functions. In addition, each class accepts different parameters and can implement various functions, based on the corresponding type. The AMSAT-IARU Link Model spreadsheet is used as a reference for calculating gain, beam-width and cutoff gain etc...

III. Numerical Results

To showcase the robustness of the proposed architecture and to demonstrate its feasibility, three scenarios are introduced, to emulate the space to ground satellite communication. We built scenes: with SDN only, without SDN, and finally SDN with streaming load. These scenarios help to see in a closer window whether this combination can improve the satellite network in terms of latency, throughput and reliability. Comparison between scenarios can also help to judge the impact of

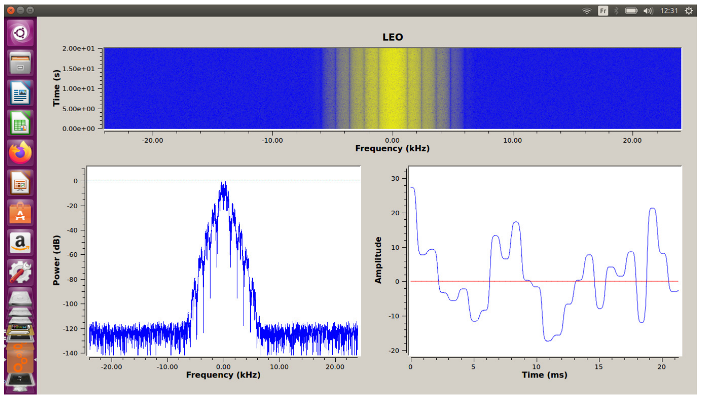
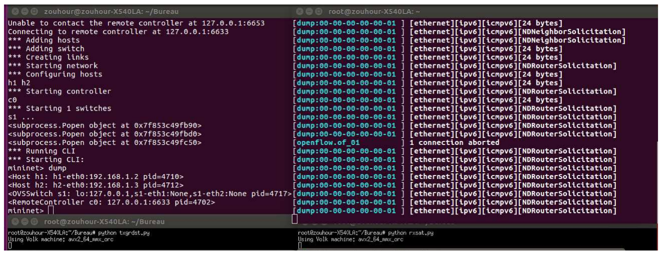


Fig. 3: Example of the developed LEO satellite transceiver using Python3 scripting.



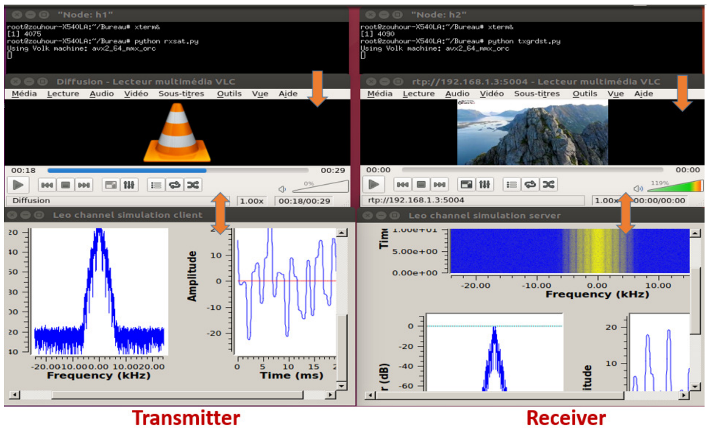
```
zouhour@zouhour-X540LA: ~/Bureau
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
<subprocess.Popen object at 0x7f853c49fb90>
<subprocess.Popen object at 0x7f853c49fbd0>
<subprocess.Popen object at 0x7f853c49fc50>
*** Running CLI
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:192.168.1.2 pid=4710>
<Host h2: h2-eth0:192.168.1.3 pid=4712>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=4717>
<RemoteController c0: 127.0.0.1:6633 pid=4702>
mininet>

root@zouhour-X540LA: ~/Bureau
root@zouhour-X540LA:~/Bureau# python txgrdst.py
Using Volk machine: avx2_64_mmx_orc
```

```
root@zouhour-X540LA: ~
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][24 bytes]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDNeighborSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDNeighborSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][24 bytes]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][24 bytes]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][24 bytes]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][24 bytes]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][24 bytes]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[openflow.of_01         ] 1 connection aborted
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]
[dump:00-00-00-00-00-01 ] [ethernet][ipv6][icmpv6][NDRouterSolicitation]

root@zouhour-X540LA:~/Bureau# python rxsat.py
Using Volk machine: avx2_64_mmx_orc
```

Fig. 4: Example of Tx(left)-Rx(right) transmission with SDN and GNU Radio enabled: the LEO satellite case.



```
"Node: h1"
root@zouhour-X540LA:~/Bureau# xterm&
[1] 4075
root@zouhour-X540LA:~/Bureau# python rxsat.py
Using Volk machine: avx2_64_mmx_orc
```

```
"Node: h2"
root@zouhour-X540LA:~/Bureau# xterm&
[1] 4090
root@zouhour-X540LA:~/Bureau# python txgrdst.py
Using Volk machine: avx2_64_mmx_orc
```

Diffusion - Lecteur multimédia VLC
Média Lecture Audio Vidéo Sous-titres Outils Vue Aide
00:18 00:29
Diffusion 1.00x 00:18/00:29
rtp://192.168.1.3:5004 - Lecteur multimédia VLC
00:00 00:00
rtp://192.168.1.3:5004 1.00x 00:00/00:00
Leo channel simulation client
Frequency (kHz)
Amplitude
Time (ms)
Leo channel simulation server
Frequency (kHz)
Transmitter
Receiver

Fig. 5: GEO/LEO satellites transceiver SDN and GNU Radio based, with successful streaming transmission

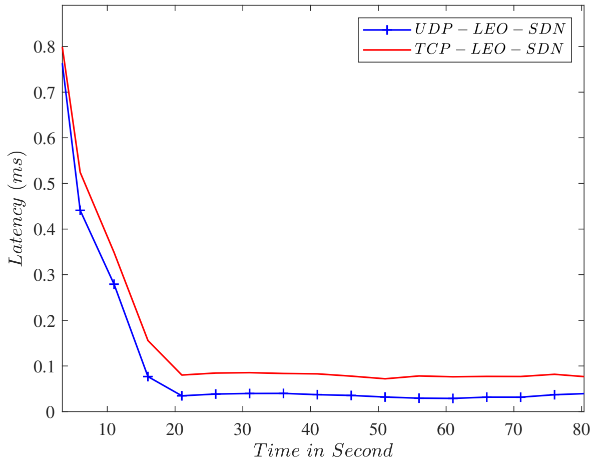


Fig. 6: Latency performance, for LEO sat, with SDN when a streaming load is enabled.

SDN and GNU radio, together on the quality of satellite communication. The satellite communication parameters are fully monitored by GNU radio blocks, such as : FEC, channel model, frequency ranges, modulations types etc. . .as illustrated in Fig. 2.

Measurements are collected using a real time script ”shell” and the ”iperf” commands under Linux operating systems between two end users. The maintained data throughout the scenarios is stored in a CSV file and treated using MATLAB software. The ”Mininet” version used in the experiment is 2.3.0d6, the operating system distribution used is ”Ubuntu 16.04”, the GNU Radio Companion version is 3.7.9, and Python 3.0 is used as a compiler for the developed APIs.

Scenario 1: *Without SDN*
Traditional terrestrial satellite links are established without SDN, which is built on capabilities envisaged for the improved satellite networks SDN/GNU Radio. This scenario will be used as the benchmark as it shows the performance of the physical layer of a GEO/LEO satellites using GNU radio emulator. All used components to ensure connectivity are illustrated in Fig. 2. Fig. 3 illustrates the successful build of a GEO/LEO satellites transceiver using Python3, before the SDN deployment. We notice the good and correct emulation of the received signal between two IP nodes (end users) as it is shown by the time domain and the frequency domain plot. This is a cross layer solution to send a satellite traffic between two IP nodes using GNU Radio satellite communication chain.

Scenario 2: *With SDN*
It focuses on performing data transmission in a satellite network with SDN as illustrated in Fig. 4 , within the satellite segment on the ground. This scenario demonstrates the integration and the implementation of GEO/LEO satellite communication chain within a SDN network model. A full Southbound API is developed to ensure the integration of GNU radio components illustrated in Fig. 2 to be deployed in an SDN architecture using POX controller.

Scenario 3: *SDN with streaming*
It is build on the capacities envisaged in scenario 2 for the SDN-GNU Radio improved satellite networks. In this use case, a streaming load is added to stress the link and to ensure when a real time networking service is required whether the link is still reliable and stable. This scenario helps to increase the level of Quality of Service-Quality of Experience (QoS-QoE) for the GEO/LEO transmission as shown in Fig. 5. While the left side shows the ongoing transmission, the right side of Fig. 5 shows the perfect reception of the transmitted video streaming.

When it comes to latency as in Fig. 6, TCP protocol shows better performance when SDN is enabled for LEO satellite. In

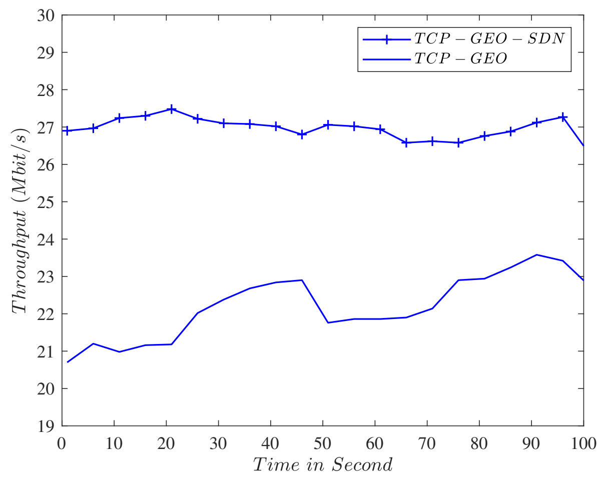


Fig. 7: Throughput performance for GEO sat, with and without SDN under TCP protocol.

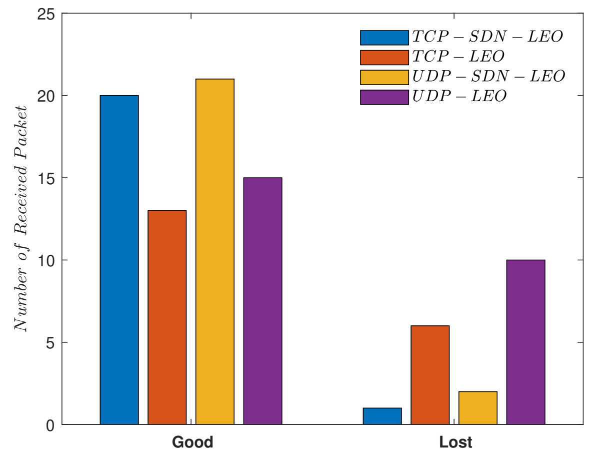


Fig. 8: Packet Loss performance for LEO sat, with and without SDN, under UDP and TCP protocols. Packet Loss statistics, ”Good”=no packet lost, ”Lost”=packet lost

this use case we added a streaming load as shown in Fig. 5 to see in a closer window the behavior of latency when the link is loaded. It can be noticed, that the time spent in terms of latency when SDN is used becomes ten times higher compared to scenario 1. This could be explained first by the processing time needed by the POX controller to send the required data to the destination. Secondly the southbound API is written in Python3 which is not a Just-In-Time (JIT) compiler.

In Fig. 7, we considered the throughput performance for GEO satellite. An improvement of approximatively 6 Mbits/s is recoded, when SDN is applied, compared to scenario 1, when GEO satellite is deployed under TCP protocol only. This could be returned to Openflow protocol for the orchestration of the allocated resources to end users. OpenFlow has an entire control of the network including resource allocation. This can help improving the balancing network environment.

In Fig. 8, we considered the packet transmission performance for GEO satellite. The number of good received packets and lost packets is recorded. Statistics shows that the number of good received packets is recorded for TCP and UDP when SDN is enabled (scenario 2). We notice that reliability and connectivity are better ensured under SDN and the controller proves again that it can maintain good connectivity.

IV. Conclusion

This paper focuses the light on the future of satellite telecommunications. In particular, the combination of SDN and GNU Radio technologies in the field of GEO/LEO satellites is considered as a starting point for making satellite communications possible when integrated into a planned SDN network architecture. Three use cases are described in this paper showing how SDN and GNU Radio technologies can help in the integration and the implementation of space to ground satellite communications. The main scenarios have been identified and analyzed, with an emphasis on improving the infrastructure of the satellite network (scenario 1) and on improving satellite offers for shared terrestrial systems such as communication services by SDN satellite (scenario 2 and 3). Multiple opportunities and benefits have been gained for each scenario, as well as new opportunities through the move from a conventional network to a SDN.

Based on the results of the presented simulations, we found that the architecture was able to obtain better quality of service thanks to an efficient control of satellite traffic under SDN and GNU Radio. However, the paper remains open to possible extensions such as high availability and the use of the P4 language which allows programming of the flow processed by the routing of packets on the transmission equipment without taking into account the format of a packet.

Acknowledgment

Authors address a special thanks to Zouhour Raddaoui, Laouini Nour Elhouda from the Higher Institute of Computer Science of Mahdia, Tunisia, and Danial Saleemi for their assistance in Python3 scripting and in the implementation process.

References

[1] Y. Gao, H. Ao, Z. Feng, W. Zhou, S. Hu, and X. Li, “Modeling and Practise of Satellite Communication Systems Using Physical Layer Security: A Survey,” in *IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2017, Guangzhou, China, July 21-24, 2017, Volume 1*, 2017, pp. 829–832.

[2] M. R. Haque, S. C. Tan, Z. Yusoff, K. Nisar, C. K. Lee, B. S. Chowdhry, S. Ali, S. K. Memona, and R. Kaspin, “SDN Architecture for UAVs and EVs using Satellite: A Hypothetical Model and New Challenges for Future,” in *18th IEEE Annual Consumer Communications & Networking Conference, CCNC 2021, Las Vegas, NV, USA, January 9-12, 2021*. IEEE, 2021, pp. 1–6.

[3] S. Singh and R. K. Jha, “A Survey on Software Defined Networking: Architecture for Next Generation Network,” *CoRR*, vol. abs/2001.10165, 2020.

[4] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzialic, “A Survey on Data Plane Flexibility and Programmability in Software-Defined Networking,” *IEEE Access*, vol. 7, pp. 47 804–47 840, 2019.

[5] L. Yan, X. Ding, and G. Zhang, “Dynamic Channel Allocation Aided Random Access for SDN-Enabled LEO satellite iot,” *J. Commun. Inf. Networks*, vol. 6, no. 2, pp. 134–141, 2021.

[6] “”OpenFlow switch specification version 1.5.0,” available at,” Ph.D. dissertation, http ://www.opennetworking.org, December., 2015.

[7] X. Hu, Y. Zhang, X. Liao, Z. Liu, W. Wang, and F. M. Ghannouchi, “Dynamic Beam Hopping Method Based on Multi-Objective Deep Reinforcement Learning for Next Generation Satellite Broadband Systems,” *IEEE Trans. Broadcast.*, vol. 66, no. 3, pp. 630–646, 2020.

[8] M. M. Aurizzi, T. Rossi, E. Raso, L. Funari, and E. Cianca, “An SDN-based traffic handover control procedure and SGD management logic for EHF satellite networks,” *Comput. Networks*, vol. 196, p. 108260, 2021.

[9] N. Torkzaban and J. S. Baras, “Controller Placement in SDN-enabled 5G Satellite-Terrestrial Networks,” *CoRR*, vol. abs/2108.09176, 2021.

[10] S. Shi, G. Li, Z. Li, B. Gao, and Z. Luo, “Dynamic Power Allocation Based on Rain Attenuation Prediction for High Throughput Broadband Satellite Systems,” *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 100-A, no. 9, pp. 2038–2043, 2017.

[11] T. Ahmed, A. Alleg, R. Ferrús, and R. Riggio, “On-Demand Network Slicing using SDN/NFV-enabled Satellite Ground Segment Systems,” in *4th IEEE Conference on Network Softwarization and Workshops, NetSoft 2018, Montreal, QC, Canada, June 25-29, 2018*. IEEE, 2018, pp. 242–246.