International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:6, 2015

Scalable Cloud-Based LEO Satellite Constellation Simulator

Karim Sobh , Khaled El-Ayat , Fady Morcos, Amr El-Kadi

Abstract—Distributed applications deployed on LEO satellites and ground stations require substantial communication between different members in a constellation to overcome the earth coverage barriers imposed by GEOs. Applications running on LEO constellations suffer the earth line-of-sight blockage effect. They need adequate lab testing before launching to space. We propose a scalable cloud-based network simulation framework to simulate problems created by the earth line-of-sight blockage. The framework utilized cloud IaaS virtual machines to simulate LEO satellites and ground stations distributed software. A factorial ANOVA statistical analysis is conducted to measure simulator overhead on overall communication performance. The results showed a very low simulator communication overhead. Consequently, the simulation framework is proposed as a candidate for testing LEO constellations with distributed software in the lab before space launch.

Keywords—LEO, Cloud Computing, Constellation, Satellite, Network Simulation, Netfilter.

I. Introduction

LOW Earth Orbits (LEO) have recently gained in popularity in the area of satellite communication compared with Geostationary orbit satellites (GEO), mainly due to their proximity to Earth. GEOs have been the traditional choice for communication satellites as they allow uninterrupted communication between the ground station and the satellite. On the other hand, GEOs are equatorial orbits; thus have poor polar coverage, and since they orbit at roughly 35,000 km above Earth, they are understandably expensive to launch, require high power for signal transmission, and incur large transmission delays.

Although LEOs sacrifice the wide coverage of GEOs, an LEO constellation allows full earths coverage, shorter communication delays, and substantially lower launch cost and lower power requirements. The inherent weakness of LEOs, however, is the limited instantaneous field of view (IFOV), high orbital velocity with short communication time windows with a ground station or receiver. To utilize LEOs for efficient communication, a satellite constellation is used, taking advantage of inter-satellite link communication to relay data among constellation members[16]. If a satellite handling a communication link moves out of the range of its control center, the link is passed on to another constellation member that has line of sight to the the control center, thus allowing uninterrupted communication. Note that line of sight connectivity is a problem between ground stations and orbiting satellites, as well as between different satellite constellation members [9][20][17]. Consequently, constellation design is a complex process that requires defining several principal factors, as detailed in Wertz [24]. The goal of the constellation designer is to minimize the cost while meeting all functional and connectivity requirements.

LEO satellites by nature host a distributed software environment. A set of computing nodes, LEO satellites, that collaborate through message exchange over a communication medium to achieve a common goal. Since it is not practical to test the constellation in space, a simulation is used to simulate the line-of-sight blockage , that will be experienced by the target distributed application running on LEO members.

The proposed simulator will be based on Cloud Computing, which is a leading computing technology domain that allows the consolidation of distributed computing resources and making them available in a utility oriented approach as a shared-as-a-service to customers. The Cloud Infrastructure as a Service (IaaS) model, based on virtualization, can be utilized to simulate an LEO constellation. With the help of the scalability features of cloud IaaS environments, large constellation of satellites and ground stations can be simulated [13][12][7][18][6][10].

An Analysis of Variance (ANOVA) factorial experiment [15][14] was used to test the factors affecting the simulator network throughput. The ANOVA is a mathematical statistical model capable of calculating the effects of a set of factors, in a system or a process, on a specific observed yield. ANOVA will help us to measure the effect of embedding the core of the proposed simulator into the constellation testbed network layer.

II. Background

A. LEO Communication

Applications deployed on LEO satellites are getting more mature and are running on top of modern operating systems like the open source Linux operating system. Hence, the need for a popular network application protocols such as TCP and UDP is growing, to be able to meet the needs and the requirements of such applications. Such protocols needed the existence of the Internet Protocol (IP) to be able to operate. In general, the trend is to enable a satellite to communicate over internet protocol to be able to transparently use ready made

Karim Sobh is with the Department of Computer Science and Engineering of the American University in Cairo (e-mail: kmsobh@aucegypt.edu).
Khaled El-Ayat is with the Department of Computer Science and Engineering of the American University in Cairo (phone: +20226152975, e-mail: kalayat@aucegypt.edu).
Fady Morcos is with the School of Sciences and Engineering of the American University in Cairo (phone: +20226153059, e-mail: fady.michel@aucegypt.edu)
Amr El-Kadi is with the Department of Computer Science and Engineering of the American University in Cairo (phone: +20226152988, e-mail: elkadi@aucegypt.edu).

applications and libraries that are based on internet application layer protocols, such as HTTP and FTP. Thus, for the proposed simulator to be of a benefit to the target environment, all communication aspects will be based on the internet protocol, mainly TCP and UDP [25]. In the related work section, we will be enumerating examples of already existing instances of constellations utilizing the internet protocol and their target missions.

B. Cloud Computing

Cloud computing is a mechanism for consolidating computing resources and sharing them as services. This concept of utility computing has been there since the mainframe was invented, but limited advances in computing resources constrained its evolution and spread. Cloud computing is all about the generic ability to consolidate any number of heterogeneous resources and share them transparently on demand. With the help of cloud middleware, a management environment for computing resources, applications and users can share computing resources in a grid-like model, where the user of the resource does not need to know the details and the mechanisms of availing the resource. The keyword here is isolation, whereby an application using a resource is guaranteed the dedication of the resource to perform a specific task.

Cloud IaaS environments, which are based primarily on virtualization, are a good deployment environment for our proposed simulation framework, where satellites and ground stations distributed software can be deployed on virtual machines. Four important factors are behind our decision of using cloud environments, which are the underlying deployment infrastructure, which are virtualization, scalability, isolation, and elasticity.

1) **Virtualization:** virtualization enables the ease of creating virtual machines on cloud environments, which correspond to constellation objects, satellites and ground stations. This allows the ability of virtual machines configuration changes based on the role. Moreover, virtual machines templates can be created with all the software needed being installed once, and then replicate as needed. Administration and deployment is made easy and flexible for changes as requirements change.
2) **Scalability:** cloud scalability allows adding and removing constellation objects by acquiring and releasing virtual resources, which will lead to better manageability and utilization of the resources used per constellation object. Basically, scalability will allow building huge constellations with the maximum utilization possible, and less administration effort for adding the corresponding resources.
3) **Isolation:** it is very critical to be able to control mission critical shared resources like network in the simulation environment, and to make sure that the bandwidth of the target network is only utilized by the simulator virtual machines, which can be achieved with cloud virtual networks.
4) **Elasticity:** Cloud environments allow for transparent expansion and shrinkage based on the resources needed, which will generally achieve better space, power, and cooling utilization.

Of course, the proposed simulation environment can be deployed on a traditional cluster, yet all of the above mentioned characteristics will not be achievable.

III. Related Work

In this section, work related to satellite constellations and their communication will be discussed. Nelson [17] presented an approach for designing a nearly constant coverage CubeSat constellation. It established a communication link between Tenby, Pembrokeshire, Wales and tactically chosen locations in the United States. The paper presents a thorough study of LEO orbits in general, and in inter-satellite communication, taking into consideration all aspects of orbit calculation methods, amateur radio antenna types, noise handling, multi-hop communication, and most importantly line-of-sight considerations. The Satellite Tool Kit software, STK, was used to design orbits and test constellations coverage, however software execution of satellite communication between different satellites of the constellation to establish constant coverage is not mentioned.

Smalarz [20] used the STK spatial and temporal analysis features in conjunction with MATLAB graphical interface SATCAT to simulate the data relay between the satellites of a CubeSat constellation. This approach targeted improving relay performance and the increase of communication duration between the constellations CubeSats and ground stations. It was shown that applying the communication relay through the constellation's satellites improves the relay time dramatically, and increases the number of accesses. It is important to note that these results were achieved by simulation. An added benefit would be an external transparent network simulator that can simulate the line-of-sight earth blockage effect.

Agogino et al. [6] introduced the idea of deploying decentralized distributed agent-based resource sharing mechanisms in CubeSat constellations. The paper proposed different types of agents that can be deployed on CubeSats that perform small tasks whose results can be shared and relayed to other agents for further processing, analogous to peer-based distributed systems. It is noted that the introduction of the line-of-sight earth blockage effect simulation would be of great value.

Muri and McNair [16] presented a comprehensive historical study of satellite constellations from 1972, such as OSCAR satellites equipped with amateur radio communication, enumerating a series of satellite projects. They also discussed the evolution of the satellite communication and inter-networking, and the introduction of Internet Protocol Layering in space communication, delay tolerance, radio frequency allocation, optical communication, and applications and orbital properties. The paper indicates the need for adopting internet protocols in space.

Challa and McNair [10] proposed a torrent like protocol. The CubeSat Torrent Protocol (CST) aims at increasing downlink and uplink speeds of large files by simultaneously downloading file portions from different CubeSats. A high

speed inter-satellite communication link of 10 Mbps is assumed to support data replication. A simulation for the data replication protocol is implemented in a custom communication simulation environment. Gains in download and upload time is improved by a factor of the constellation size. However, line-of-sight earth blockage effect is marginally addressed via peers network probing.

An overview of the IRIDIUM constellation for global mobile phone coverage is presented by Pratt et al. [19] and Muri et al. [17]. The LEO IRIDIUM constellation constitutes 66 large satellites arranged in 11 planes. Each satellite maintains up to four inter-satellite links to its neighbours. Members of the IRIDIUM constellation are satellites, gateways, and users handsets. A satellite in direct contact with initiating handsets acts as a gateway, and as a repeater routing phone call traffic. Ground stations are used as a bridge to the public telephone system, making it possible to route a call between two handsets through an IRIDIUM satellite.

QB50 is a good example of a CubeSat Constellation that will not benefit from the proposed simulation framework. Bedon et al. [8] proposed two constellation topologies for QB50, which were simulated and tested with respect to the application protocol used for communication, namely TCP versus UDP. In the two topologies all the CubeSats in the QB50 constellation are moving in the same orbit. The first topology is a ring of 50 CubeSats with an equal distance separating them, while the second topology is a string of CubeSats 10,000 KM long making the constellation look like a train rather than a ring. The CubeSats in QB50 constellation communicate with neighbors, successor and predecessor CubeSats. Data travels through the constellation until it reaches the closest CubeSat to the destination ground station. In that sense, the line-of-sight earth blockage effect has no significance on the constellation.

IV. Motivation and Problem Definition

Recent proposed LEO constellations require the collaboration of member satellites and ground stations to share mission tasks and objectives. This can be realized by the deployment of distributed software on-board the satellites, and using inter-satellite link communication and data relay for execution and processing of mission plan [16][19][21]. Our motivation is to build a low level scalable network simulator capable of simulating line-of-sight earth blockage effects on members of the constellation, using the Internet Protocol (IP) as their communication backbone.

The main problem being tackled is that distributed applications running on a constellation need to account for all communication failures and outages between different constellation members. A transparent low network overhead simulator is proposed to simulate all network outages resulting from the earth line-of-sight blockages, making them appear to the constellation objects as realistic as possible. Moreover, the target distributed application should not be aware of the existence of the network simulator, and need to be deployed on the simulation environment without the need for any code updates or amendments to suit the simulation environment.

V. Proposed Simulation Framework

A. Concept

The idea is to build a cloud environment based on the IaaS model to simulate LEO constellations. Each object in the target constellation environment will be represented by a virtual machine in the cloud environment, and all of the virtual machines will be connected via a virtual network. Three types of objects exist in our environment: Satellite, Ground Station, and Earth.

Objects of type satellite are represented by virtual machines and can communicate with other objects of either type; satellite or ground station. A constellation is built up of more than one constellation object, satellite or ground station, without any upper bound on the number of objects. Objects of type ground station are represented by virtual machines, and can communicate with other constellation objects as well.

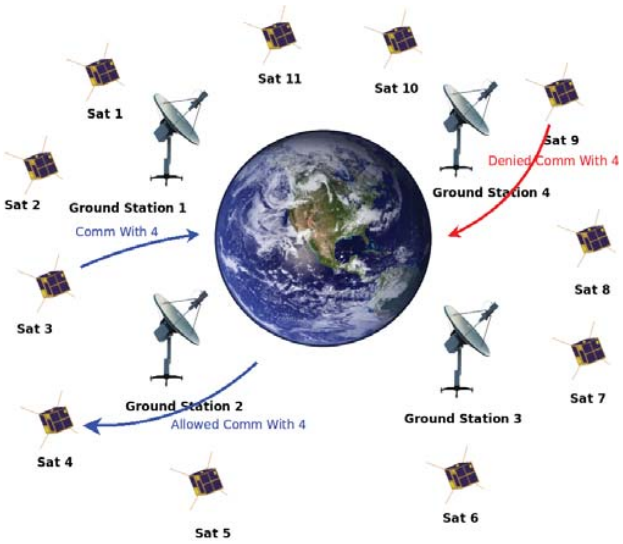

Fig. 1: Simulator Overall View

Fig. 1 illustrates the overall view of the simulation environment and the communication mechanisms between constellation members having the earth as their gateway. Only one instance of the Earth object must exist in the constellation environment, and is represented by a dedicated virtual machine. The Earth virtual machine will hold the real simulation of the traffic. The earth VM is responsible for simulating the line-of-sight between the different constellation objects. The virtual machine that will hold the earth object will host a Netfilter router [23][11] that will enable routing traffic between any two constellation objects. Any two objects that need to communicate are configured to use the earth object as a gateway, without communicating directly; no two objects of type satellite or ground station can communicate directly without routing through the earth object.

The SGP4 algorithm [2][22] will be used to calculate the locations of each object in the constellation based on their two-line elements. Based on the calculation, the line-of-sight between any 2 objects that need to communicate will be calculated; in case the two communicating objects have a clear

line-of-sight, the earth router will allow the flow of the traffic between them, else it will deny it to simulate the blockage effect of the earth between the two objects. The blockage effect can be observed in Fig. 1, where the red arrow represents denied communication between Sat9 and Sat4 due to the earth blockage effect. On the other hand, the blue arrows represent the line-of-sight clearance between Sat3 and Sat4.

B. Overall View

The simulator is built up of a number of subsystems. Fig. 2 shows the different subsystems of the simulator as well as their high level interaction to achieve a common goal, taking into consideration the earth's line-of-sight blockage effect. At the core of the simulator, the line-of-sight daemon resides to communicate with the back-end database, and serve other simulator components. The line-of-sight daemon is the most important component of the simulator, and it is deployed on the Earth virtual machine. The other simulator components, described in Fig. 2, are the back-end database, Cloud Integration Module, SGP4 Integration Module, Netfilter Linux Kernel Extension [23][11], Cesium-Based GUI Simulator [3], and the Web Administration Console.

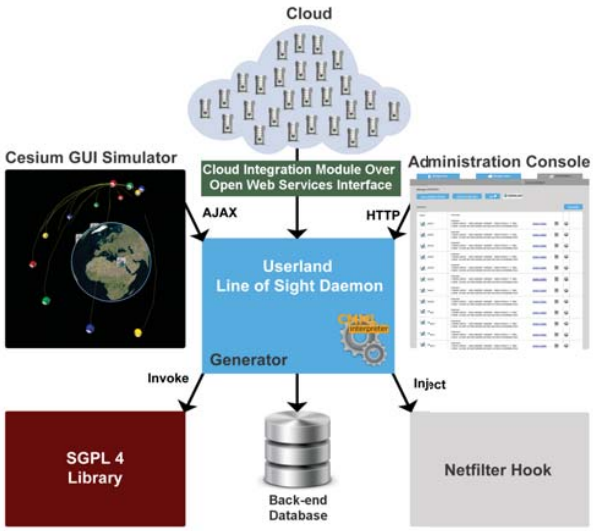

Fig. 2: Simulator Main Subsystems

C. Simulator Subsystems

1) Constellation Back-end Database: Constellation configuration data is stored in a back-end MySQL [4] database. A constellation is built up of satellites and ground stations. Satellites are defined by their two-line elements, from which their location at any point in time can be calculated. Ground stations are also members of constellations, and are defined by their Cartesian attributes. Constellation satellites and ground stations are mapped to cloud virtual machines, together with their network attributes definition. All of this is stored in the back-end relational database.

2) Cloud Integration Module: The cloud integration module is used to import cloud virtual machines configuration and network attributes into the simulator back-end database. This allows linking virtual machines to different satellites and ground stations. The integration module import data from the cloud middleware through open web service interfaces.

3) SGP4 Integration Module: The SGP4 is an open source C++ library that calculates the Cartesian attributes of a satellite based on its two-line elements and time. This module instantiates SGP4 objects, and passes the two line elements of the satellites to it to calculate and store their Cartesian attributes in an array of attributes, together with the ground stations Cartesian attributes.

4) Line-of-Sight Daemon: This is one of two main core components of the simulator. The daemon is responsible for calculating a line-of-sight matrix between all satellites and ground stations in a constellation. The line-of-sight daemon generates a matrix of numbers representing the distance between each satellite and/or ground station in the target constellation. A cell in the matrix that has a distance of -1 indicates that the earth is in the way. Each constellation has its own line-of-sight matrix. The daemon will be catering to both the Linux Netfilter routing engine extension and the Cesium GUI simulator. The daemon injects the line-of-sight matrix into the Netfilter Kernel extension character device buffer, moving the data from user space to kernel space. Two web service interfaces are in place to be utilized by the Cesium GUI front-end APIs, to fetch satellite locations and their movements. The administration console will also utilize the web services interfaces to display the line-of-sight matrices for monitoring purposes by administrators.

5) Netfilter Kernel Module Extension: A Netfilter Linux Kernel Module Extension is at the heart of the simulator. The module has a character device that is considered the data entry point. The line-of-sight daemon injects the line-of-sight matrix to the Netfilter hook character device periodically. The Netfilter hook is basically a hook into the Network IP layer. All packets going through the network stack are intercepted and passed to the hook routine before routing decisions and further network packet processing. The hook extracts the source and destination addresses of each network packet and looks them up from the line-of-sight matrix. If any of the packet's source or destination are not found in the matrix, the packet is passed to the higher network stack level. On the other hand, if the source and the destination are found in the line-of-sight matrix, the corresponding value representing the distance between them is checked, and if the value is greater than 0 the packet is passed to the higher network stack level, otherwise the packet is dropped.

6) Cesium GUI APIs Front-end Integration: Cesium [3] is a client side javascript based framework for simulating the earth and objects rotating around it, as well as locations on the earth. An integration interface based on Ajax (Asynchronous JavaScript and XML) is built to allow the simulator to fetch satellite locations from the line-of-sight daemon. The client side application probes the line-of-sight daemon web service interfaces iteratively and periodically, and the retrieved coordinates are used to move the satellites in their orbits

around the earth.

7) Administration Console: The administration console is a web application used to create constellations and launch the Cesium simulator GUI. It also allows the administrator to inspect the line-of-sight matrices by initiating Ajax calls to the line-of-sight daemon. Moreover, the administrator is capable of mapping cloud virtual machines to constellation objects. The administration console is a traditional PHP/MySQL web application deployed on an Apache web server [1].

D. Framework Workflow

The simulator environment is deployed over a cloud infrastructure. A private IaaS, Infrastructure-as-a-Service, cloud model is used to serve the constellation simulation. Using IaaS, we are able to use virtual machines to represent satellites, ground stations, and the earth. Each virtual machine is a completely independent entity with its own OS running on it. A cloud virtual network is in place to allow communication between virtual machines.

The Netfilter simulator lies at the core of the environment, and is deployed on the earth virtual machine. As described earlier, the configuration of satellites and ground stations locations, as well as their linkage to the cloud virtual machines is done using the administrative console prior to starting simulation.

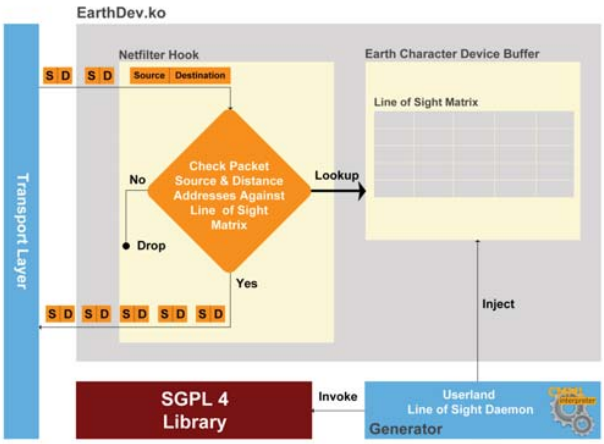

Fig. 3: Simulator Workflow

The internal details of the workflow of the simulator is illustrated in Fig. 3. A userland process daemon is in place to generate the line-of-sight matrix. The userland daemon fetches the two-line elements of the satellites and the cartesian coordinates of the ground stations of the running constellations from the back-end MySQL database, and with the help of the SGP4 library the locations of the satellites and the ground stations are being calculated, based on a time parameter passed to the SGP4 library together with the two-line elements. A line-of-sight matrix is generated based on a line-of-sight algorithm that calculates the visibility between each two objects in the constellation. The cell in the matrix corresponding to any two objects in the constellation is set to the distance calculated, if there is a clear line-of-sight between them, or else is set to -1. Intuitively, the diagonal of the matrix will have the values 0. This operation is performed iteratively based on a sampling rate configuration value of the target constellation, which is defined via the administration console. The matrix is generated and saved in the internal buffer of the daemon. The daemon then injects the matrix every time it is generated into the character device of the Netfilter kernel module extension, EarthDev.ko . The daemon also responds with the matrix data and the locations data to other clients, over web services protocol, such as the Cesium GUI simulator and the administration console.

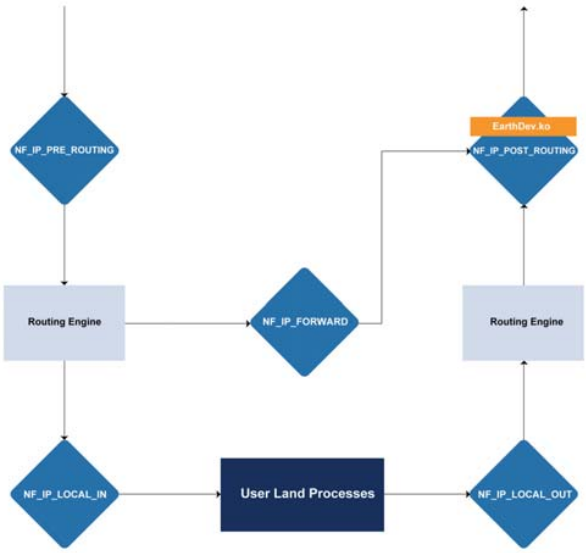

Fig. 4: Netfilter Block Diagram with EarthDev.ko Hook

Fig. 4 presents a block diagram for the internal routing phases within the Linux kernel. The Netfilter kernel module extension EarthDev.ko, installs a hook in Network Layer-3, specifically in the NF_IP_POST_ROUTING after all routing operations are performed, to intercept all packets received by earth for routing. As per Fig. 3, the module extracts the source and the destination addresses from the incoming packets and looks them up in the line-of-sight matrix. The decision of allowing the packets or blocking them is based on the existence of both source and destination addresses in the matrix, and that both have a distance value larger than 0. If either the source or the destination addresses are not found in the line-of-sight matrix, the packet is allowed.

Extensive locking and synchronization take place within the kernel module to synchronize between copying data from the userland daemon to the character device buffer, as well as reading from the line-of-sight matrix for packet routing decisions. It should be noted that timings within the kernel is crucial as the whole process of looking up entries in the line-of-sight matrix should be performed most efficiently.

E. Framework Deployment

The simulation environment is scalable and cloud enabled, which allows for flexible deployment. Fig. 5 presents the deployment diagram of the simulator subsystems. The device boxes presents the boundaries of a virtual machine or a physical node. A device box can host different simulator subsystems. All of the subsystems boxes can be deployed on cloud IaaS virtual machines. The deployment diagram in Fig. 5 shows the affinity of certain components, meaning that all the software modules that are deployed within the boundaries of the same device box will need to be physically deployed within the same virtual or physical environment; basically, deployed within the same operating system boundaries.

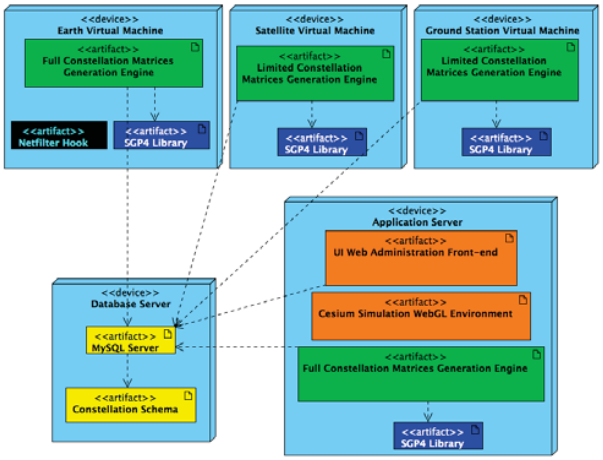

Fig. 5: Framework Deployment Diagram

The SGP4 library is needed on all node types except the database server, and it is essential to exist on the earth virtual machine and the application server. SGP4 is optional for the applications running on the satellites and the ground stations, if position related data is required. The Netfilter hook extension is deployed on the earth virtual machine for packet routing. The Cesium javascript library and the front-end administrative web application are deployed on the application server. It is very important to highlight that the only operating system constraint is that the earth VM needs to have a Linux OS to be able to install the Netfilter hook extension. On the other hand, all the constellation objects virtual machines need to have an OS that can support the internet protocol. Hence, the simulator can simulate the line-of-sight blockage effect for any satellite application running on any operating system environment.

For performance scaling the MySQL server can be replicated through a MySQL replication cluster for better performance throughput and fault tolerance. The GUI application server for Cesium simulation and the web administration console can also be scaled in a cascaded manner and deployed behind a dispatcher cluster.

VI. Experiments and Results

To asses the simulation framework, the Netfilter Module extension effect on the overall network transport performance is measured. The Analysis of Variance (ANOVA) factorial experiment design [15][14] was used to model and simulate the execution of the earth virtual machine. Performance is measured with respect to traffic initiated from satellites and ground stations virtual machines. We used the R language [5] to run the simulation and present the results.

ANOVA factorial experiment design is a statistical modelling approach that can identify influential factors on a specific process or system. In our experiment, we define three factors to investigate their effect on network throughput. The three factors are stream size, number of communicating satellites, and the deployment of the line-of-sight matrix together with the Netfilter kernel module extension.

The stream size is the size of the message being sent through the earth gateway. The number of communicating satellites represents the number of concurrent threads of communication at any point in time. The Netfilter kernel module extension is plugged into the kernel, and with the help of the line-of-sight daemon the line-of-sight matrix is injected to the Netfilter module extension.

A. Experiment Description

Our main objective is to measure the effect initiated by searching the line-of-sight matrix on per packet transmission between any two VMs through the earth gateway router. As the number of communicating VMs gets bigger, two overhead effects will be incurred. First, the number of concurrent connections going through the earth virtual machine will be larger, imposing communication overhead due to the multiplexing of forwarding packets between different concurrent streams. The second overhead results from the larger search space presented by the large line-of-sight matrix, which is a function of the number of communicating satellites and ground stations.

A client application deployed on satellites and ground stations virtual machines is used to generate results that is used as the input to the ANOVA model. The application is a simple network round trip calculator. The application is split into two parts running in concurrent threads, client and server. The client part is capable of sending a network message, built up of a sequence of network packets, to any other application running on other virtual machines.

The client can send concurrent messages, with a specific size based on a configuration file. The server thread waits for a connection, and upon receiving a connection it dispatches a separate thread to serve it, allowing for serving concurrent connections. The server thread's job is to reply to its corresponding client with the same message and close the connection. The client verifies the received message and makes sure that it is identical to the original one. The messages are generated randomly as per each individual connection to make sure of the validity of the message after the round trip travel. The client records a time stamp before sending the packet and upon receiving the reply, and then calculates the round trip time for the sending of the packet back and forth. The time and the rest of the request attributes are then saved to a log file.

ANOVA Results - Before Transformation

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
stream_size	1.000000	2322537.852192	2322537.852192	958.837564	0.000000
threads	1.000000	6427000.572559	6427000.572559	2653.325787	0.000000
earth_dev	1.000000	158517.737070	158517.737070	65.442533	0.000000
stream_size:threads	1.000000	1454079.847925	1454079.847925	600.302974	0.000000
stream_size:earth_dev	1.000000	341.476154	341.476154	0.140975	0.707353
threads:earth_dev	1.000000	39373.846974	39373.846974	16.255117	0.000057
stream_size:threads:earth_dev	1.000000	58.827571	58.827571	0.024286	0.876174
Residuals	1992.000000	4825108.624410	2422.243285		

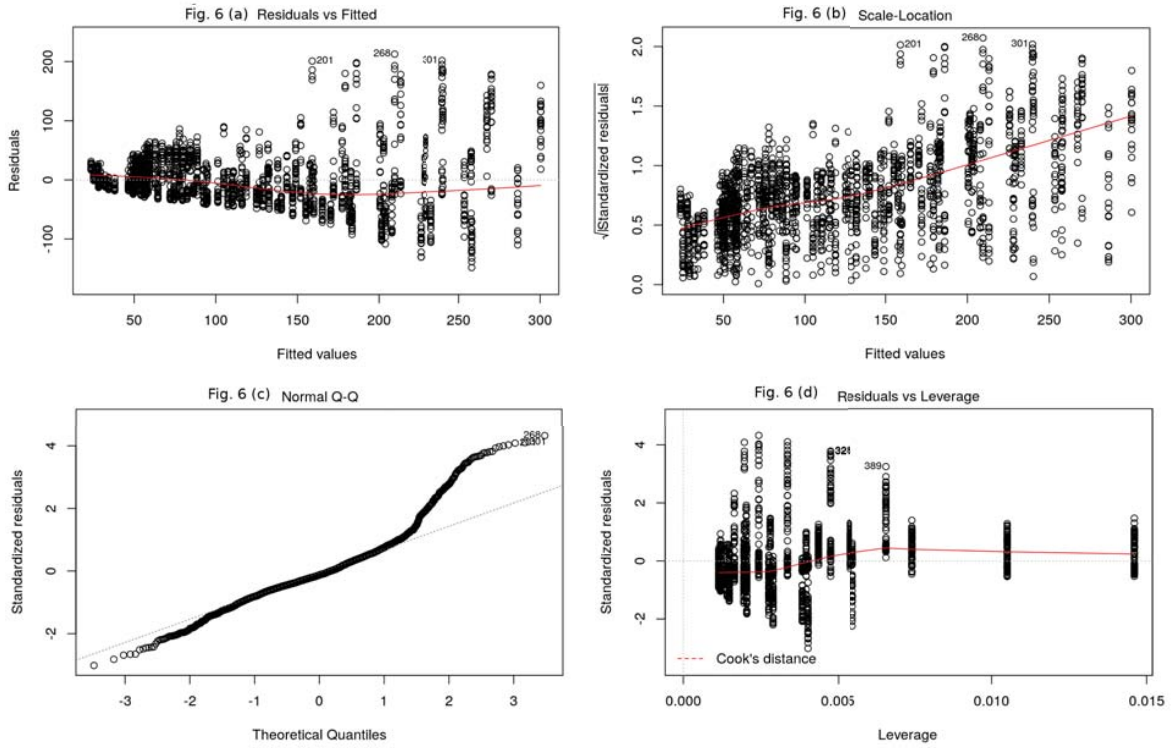

Fig. 6: ANOVA Table and Graph Plots Before Normalization

In the experiment runs, 20 virtual machines are used to represent the communicating satellites and ground stations. We have performed two main experiment runs, which are identical in everything except for the activation of the earth gateway firewall where the line-of-sight matrix daemon is activated. In the first run, the native Linux firewall router is configured on the earth VM to allow forwarding network packets from any source to any destination. While in the second experiment run our kernel extension is loaded with the line-of-sight matrix, to be able to measure the effect of the simulator extension on the overall network transport performance. It is important to mention here that for the sake of the experiment validity, we have modified the line-of-sight userland daemon to generate a matrix with all its cells set to 1, to allow all traffic between all satellites. This modification will make sure that all packets will arrive to their destinations, as the main objective here is to measure the throughput, and not to test the validity of the line-of-sight matrix generation.

Each experiment run is divided into 50 sub-runs. The amount of traffic to be transferred in each sub-run is set to 500 MB, which was chosen to be large enough and to take a long duration to absorb any network fluctuations or outliers, which usually occur at the start of launching the communicating threads. The message size in the first sub-run is set to 10 KB, with 2 concurrent threads; which means that each client will send a 10 KB message to 2 clients in parallel. The sending client keeps all the recipient clients in a list, and loops through them 2 by 2 until the list is exhausted. The client then will

ANOVA Results - After Box-Cox Transformation

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
stream_size	1.000000	280.942402	280.942402	912.017383	0.000000
threads	1.000000	1460.240648	1460.240648	4740.348352	0.000000
earth_dev	1.000000	30.328599	30.328599	98.455091	0.000000
stream_size:threads	1.000000	106.901621	106.901621	347.032474	0.000000
stream_size:earth_dev	1.000000	0.707431	0.707431	2.296520	0.129823
threads:earth_dev	1.000000	1.126514	1.126514	3.656980	0.055978
stream_size:threads:earth_dev	1.000000	1.962157	1.962157	6.369708	0.011686
Residuals	1992.000000	613.625657	0.308045		

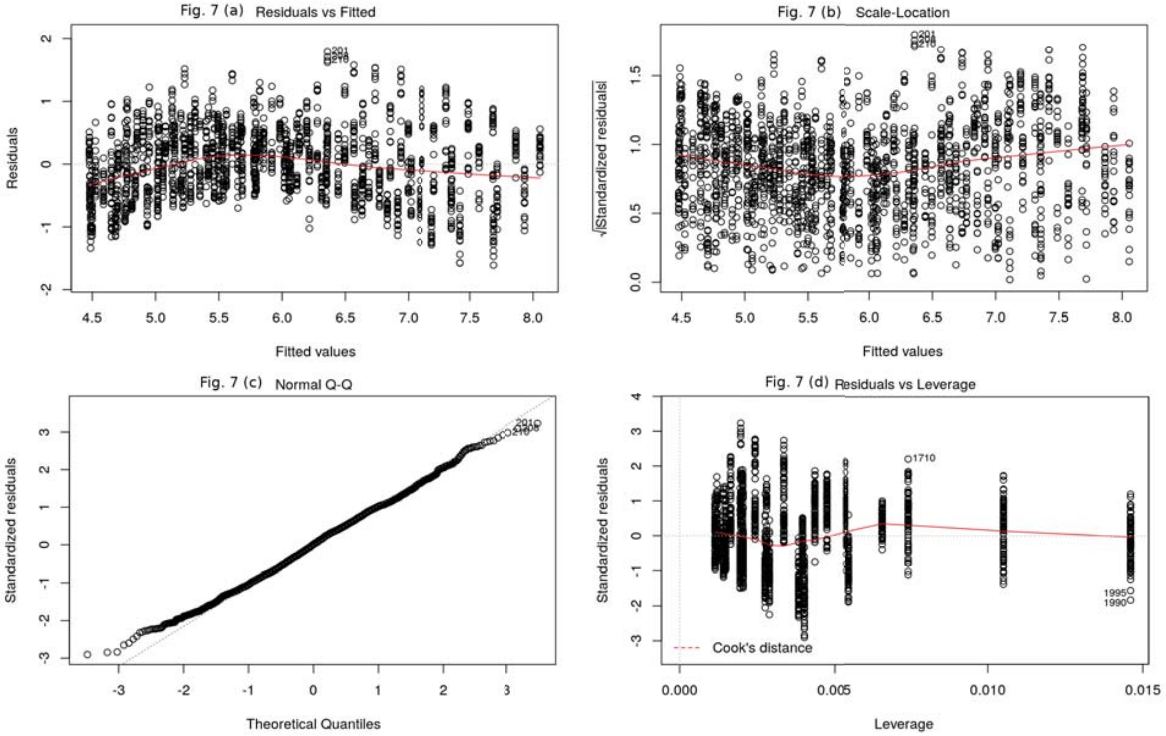

Fig. 7: ANOVA Table and Graph Plots After Normalization

need to start over from the beginning of the list. In subsequent runs the number of the concurrent threads is incremented by 2 until 20 concurrent threads is reached, which is the number of communicating virtual machines in the constellation; this sums up to 10 sub-runs. The 10 sub-runs are repeated after doubling the message size. The message size is doubled until 200 KB, which makes the total number of sub-runs per experiment is 50 sub-runs. The total number of connections per sub-run is not fixed over all sub-runs, as they have to be calculated in a way that the total size of data being transferred is 500 MB.

The maximum number of concurrent threads going through the Netfilter kernel extension is 400 concurrent threads, and the maximum concurrent sessions traffic volume is 78 MB. The total number of log entries generated by the round trip application in the two experiments is very close to 2 million log entries. The data is stored in a MySQL database for further processing and preparation before introducing it to ANOVA. A log entry would typically have the following attributes:

1) **Run Name:** each sub-run is given a unique name to be able to identify which sub-run a log entry belongs to.
2) **Source Host Name:** the network host name of the virtual machine sending the messages.
3) **Destination Host Name:** the network host name of the virtual machine receiving the message.
4) **Message Size:** The size of the message in bytes.
5) **Threads:** the number of concurrent threads initiated by

the client in parallel with this thread.

6) **Time:** the time in seconds for the whole packet round trip transmission.

B. ANOVA Results and Interpretation

A factorial ANOVA model is run on the collected data, and the ANOVA table and plots in Fig. 6 present the initial results. One of the most important prerequisites for ANOVA to generate correct results is that the experiment data introduced to the model needs to be homogeneous and should be normally distributed. A set of graphs are generated by the ANOVA model to check the normality of the data, as well as its homogeneity. Moreover, a number of tests can be performed to numerically test the normality and the homogeneity.

As per the ANOVA table in Fig. 6, each factor in the experiment is listed in a row. ANOVA is based on checking the validity of a null hypothesis. The null hypothesis indicates that all the factors have the same effect. The P-Value located in the last column of the ANOVA table in Fig. 6 represents the probability of the corresponding factor violating the null hypothesis, which indicates weather the corresponding factor is influential or not. The P-Value needs to be compared to a threshold that is called α, and if the P-Value is less than α then the corresponding factor is considered influential. A default value for α = 0.05 is preselected, and can be changed based on specific case requirements based on prior knowledge of the problem nature.

Test	**P-Value ($<$0.05)**
Bartlett	2.2e-16
Anderson-Darling	5.458e-10
Shapio-Wilk	1.472e-10

TABLE I: HOMOGENEITY AND NORMALITY TEST RESULTS

The ANOVA model need to be checked and the plots need to be interpreted before analysing the ANOVA table. Plot A and plot B in Fig. 6 illustrate the non-homogeneity of the data, and this is very clear as data is not equally distributed over the whole graph area. Plot C illustrates that the data is not normally distributed, not showing a continuous firm straight line. Finally, plot D helps identifying outliers presented by far stand out data points, that might need further investigation, which does not show in our case. Consequently, we have applied the Bartlett test for homogeneity, and the Anderson-Darling as well as the Shapro-Wilk normality tests. The table in Fig. 8 presents the results of the tests which emphasis the negative normality and the negative homogeneity results we achieved from the graphs.

In this case, normality transformation of the data needs to be applied to be able to trust the results generated. We have applied the Box Cox transformation method on the data, and we have ran the model again on the transformed data. The ANOVA table and the 4 plots in Fig. 7 are the results of the ANOVA model after the Box Cox transformation.

As can be seen from the plots after the transformation, plot A and plot B show more data distribution over the graph area, and plot C shows a more continuous firm straight line, and hence normality and homogeneity are achieved after data transformation. This paves the way to start interpreting the ANOVA results, identify the influential factors, and determine the influence ratios between factor.

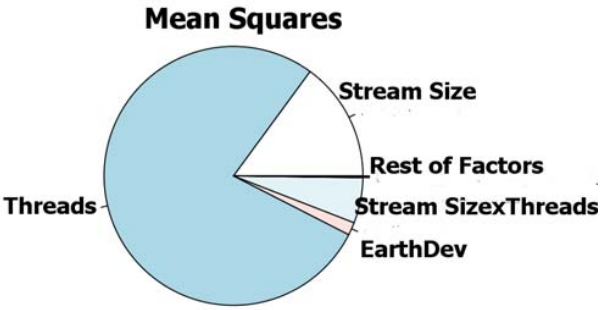

Fig. 8: Factors Mean Squares Ratios

The influential factors can be identified from their P-Values being less than α=0.05, which rejects the null hypothesis that all factors have the same means, and hence the same ratio of influence. Based on that criteria, and as can be seen from the ANOVA table in Fig. 7, the influential factors are the stream size, concurrent threads, and the existence of the earthdev kernel module extension. Moreover, the ANOVA model can detect the effect of the interaction of any set of factors, and as we can see the interaction between the stream size and the number of concurrent threads is influential. Also the interaction of the three factors is influential as well. A very important thing to mention here is that after the Box Cox transformation of the data, the interaction between the earthdev and the concurrent threads is eliminated as its P-Value raised above 0.05, which emphasis the importance of the model check we have applied.

Regression Analysis

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-6.6755402778	5.0896508963	-1.3115909939	0.1898093179
stream_size	0.0003157588	0.0000601861	5.2463781272	0.0000001716
threads	14.3963021465	0.4101357571	35.1013094979	0.0000000000
earth_dev	1.0058430556	7.1978533253	0.1397420884	0.8888778979
stream_size:threads	-0.0000834900	0.0000048499	-17.2146843861	0.0000000000
stream_size:earth_dev	-0.0000030360	0.0000851159	-0.0356690273	0.9715498413
threads:earth_dev	1.6126248737	0.5800195501	2.7802939977	0.0054820332
stream_size:threads:earth_dev	-0.0000010689	0.0000068588	-0.1558409472	0.8761741841

Regression Summary:
R Squared: 0.6831219
Adj. R Squared: 0.6820084
Root Mean Square Error: 49.21629

Fig. 9: Regression Analysis

The ratios between the effect of different factors is presented in the Pie chart of Fig. 9. As we can see, the largest effect is attributed to the number of concurrent threads, followed by the stream size, and the interaction between the stream size and the number of concurrent threads. The deployment of the earthdev extension module comes later with a very small effect

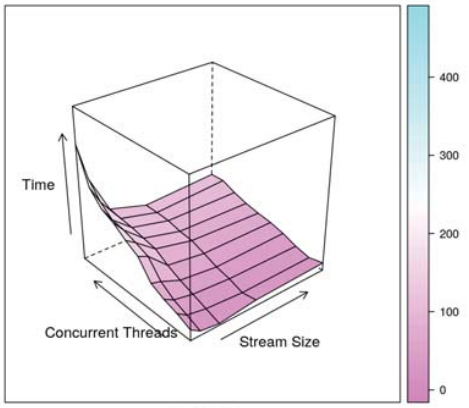

Fig. 10 (a)

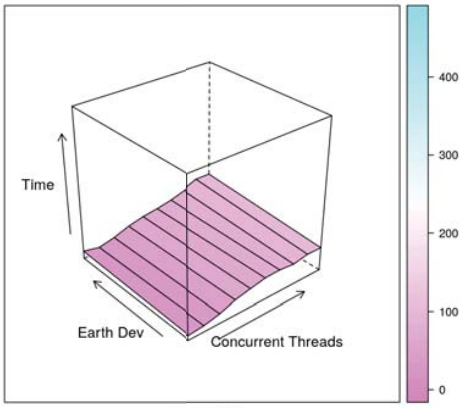

Fig. 10 (b)

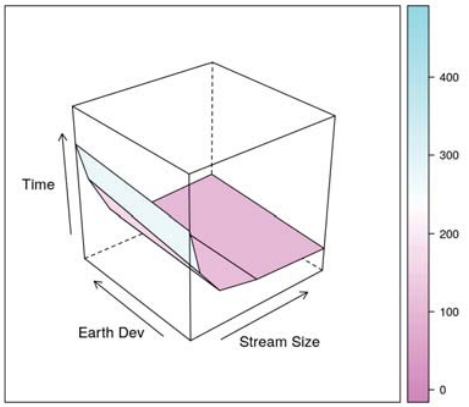

Fig. 10 (c)

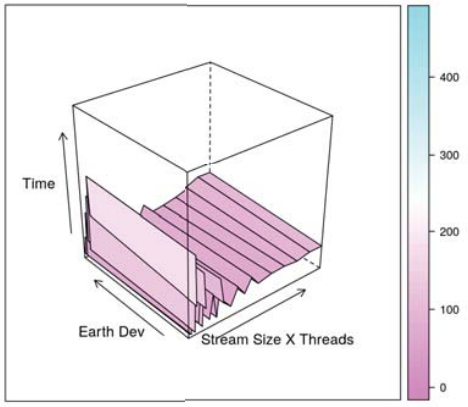

Fig. 10 (d)

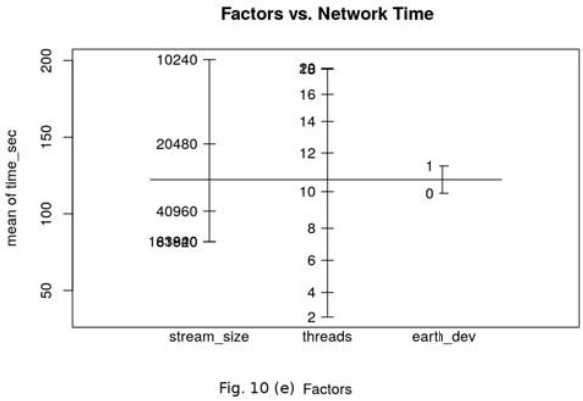

Fig. 10 (e) Factors

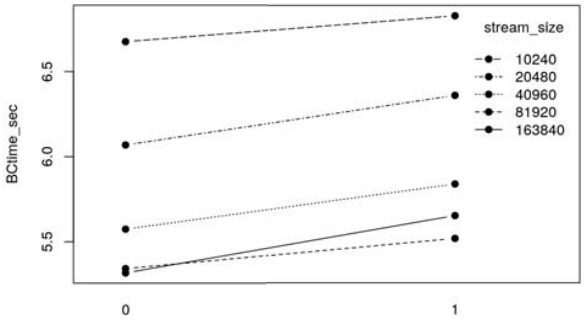

Fig. 10 (f) earth_dev

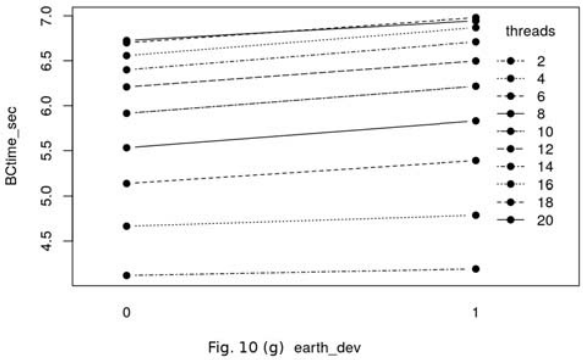

Fig. 10 (g) earth_dev

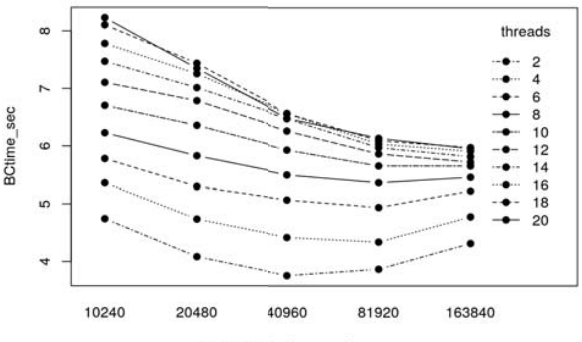

Fig. 10 (h) stream_size

Fig. 10: 3D and 2D Factors Interactions

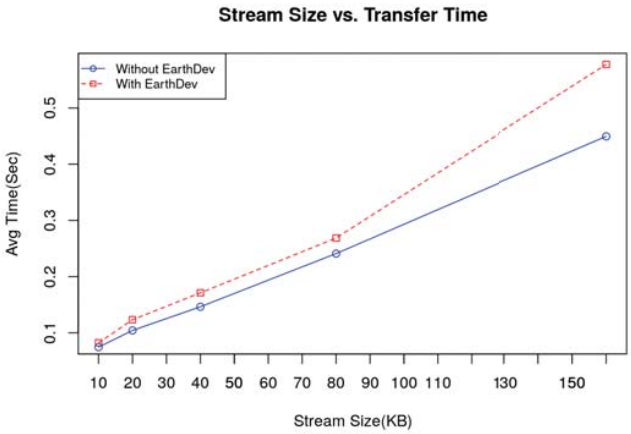

Fig. 11: Stream Size vs. Time

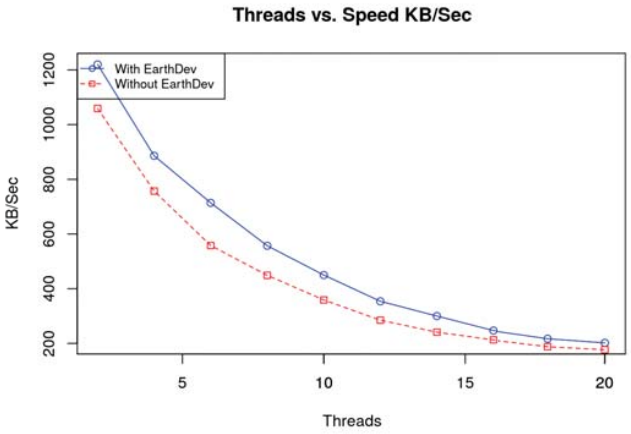

Fig. 12: Threads vs. Speed

on the yield, which implies the very low overhead imposed by the simulator kernel model extension.

The 3D and 2D interaction plots in Fig. 11 show the interaction effect of different factors. It is very obvious from the plots that the effect of the earthdev module extension is very minimal, and that the change in performance is very much attributed to both, the stream size and the number of concurrent threads.

Based on the ANOVA model applied, we were able to come up with a regression relationship between the different factors. The regression table in Fig. 10 presents the coefficients of each factor. The importance of this regression analysis is basically the ability to predict an estimate of the yield for new combinations of the studied factors that are not being covered in the experiment; for example the case of 7 concurrent threads and a message size of 15.7 KB.

To sum up, the ANOVA model has showed that our simulator module extension is of a very low overhead and that the delay in packets delivery attributed to the deployment of the simulator kernel module extension is very minimal. The two line graph plots in Fig. 12 and Fig. 13 show that the effect on the packet transport speed imposed by the kernel module extension is of a minimal effect.

VII. Future Work

Extensions and enhancements can be added to the proposed simulation framework. Two main extensions are worth mentioning, interconnected earth virtual machines, and traffic shaping.

For scalability, if the number of constellation objects gets bigger, the earth virtual machines can become a communication bottleneck. In that case, more than one earth virtual machine can exist, and the constellation objects can be divided into groups, each can be assigned to an earth virtual machine. The constellation objects belonging to the same group will communicate through their earth virtual machine, and the earth virtual machines will collaborate to transport traffic across the groups. This will split the traffic of the constellation, and will divide the flow of the communication to be split over more than one gateway. In that case, a virtual network connecting the earth VMs, and a virtual network for each group of constellation objects need to be in place. Consequently, the earth virtual machine will need to have two virtual network cards to connect it to the two virtual networks. We can visualize this as building a cluster of earth virtual machines which appear to the outside world, the constellation objects, as one entity, while it is built up of multiple virtual machines. Better levels of scalability can be achieved that way through adding more earth virtual machines as the target constellation size gets larger. We can go further to make the groups dynamic, meaning that constellation objects will leave and join groups based on their relative location to each other.

Traffic shaping is a way to change the speed of the communication medium between two communicating entities. This can be of much benefit if implemented based on the distance between the communicating constellation objects, which will achieve a much realistic environment. The kind of challenges that we are going to face here are related to problems resulting from inducing delays within the Linux kernel, as this might affect the internal state of the kernel, and hence might result in unexpected results. This will impose an extensive use of the Linux kernel work queue to overcome delay related problems.

VIII. Conclusion

A scalable cloud-based simulation framework is proposed for LEO constellations distributed software development. The proposed simulator framework is capable of simulating the line-of-sight earth blockage effect without updating the software application deployed on the satellites and ground stations of a target constellation. This will allow testing LEO distributed software in the lab transparently; the deployed LEO software is not aware that it is running in a simulated environment. We have conducted an ANOVA factorial experiment to investigate the performance effect on the communication layer, and the simulator showed a very low

overhead effect on the communication overall performance. As this work represents the base and the core of the cloud-based LEO simulator, we can build extra functionalities on top of it, such as interconnected earth gateways and traffic shapers. Finally, the proposed framework can be used as a test bed for testing LEO applications from the communication point of view before launching them to space.

REFERENCES

[1] Apache Website. http://www.apache.org/, 2013. (Online; accessed 15-August-2013).
[2] C++ SGP4 Satellite Library. http://www.danrw.com/sgp4/, 2013. (Online; accessed 15-August-2013).
[3] Cesium WebGL Virtual Globe and Map Engine. http://cesium.agi.com/, 2013. (Online; accessed 15-August-2013).
[4] MySQL Website. http://www.mysql.com, 2013. (Online; accessed 15-August-2013).
[5] R Language Project for Statistical Computing. http://www.r-project.org/, 2013. (Online; accessed 20-September-2013).
[6] A. Agogino, C. HolmesParker, and K. Tumer. Evolving distributed resource sharing for cubesat constellations. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 1015–1022, New York, NY, USA, 2012. ACM.
[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.
[8] H. Bedon, C. Negron, J. Llantoy, C. Nieto, and C. Asma. Preliminary internetworking simulation of the qb50 cubesat constellation. In *Communications (LATINCOM), 2010 IEEE Latin-American Conference on*, pages 1–6, 2010.
[9] J. Canales, G. Rodriguez, J. Estela, and N. Krishnamurthy. Design of a peruvian small satellite network. In *Aerospace Conference, 2010 IEEE*, pages 1–8, 2010.
[10] O. Challa and J. McNair. Cubesat torrent: Torrent like distributed communications for cubesat satellite clusters. In *MILITARY COMMUNICATIONS CONFERENCE, 2012 - MILCOM 2012*, pages 1–6, 2012.
[11] J. Corbet, A. Rubini, and G. Kroah-Hartman. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005.
[12] D. Hilley and D. Hilley. Cloud computing: A taxonomy of platform and infrastructure-level offerings, April 2009.
[13] C. N. Hoefer and G. Karagiannis. Taxonomy of cloud computing services. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 1345–1350, 2010.
[14] R. Jain. *The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley, 1991.
[15] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2006.
[16] P. Muri and J. McNair. A survey of communication sub-systems for intersatellite linked systems and cubesat missions. *Journal of Communications*, 7(4), 2012.
[17] J. M. NELSON. *Persistent Military Satellite Communications Coverage Using A CubeSat Constellation In Low Earth Orbit*. PhD thesis, University of Central Florida Orlando, Florida, 2010.
[18] B. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pages 44–51, 2009.
[19] G. B. Shaw. The generalized information network analysis methodology for distributed satellite systems. Technical report, Doctor of Science Thesis, MIT, 1998.
[20] B. R. Smalarz. Cubesat constellation analysis for data relaying. 2011.
[21] J. E. Underwood. Distributed satellite communication system design: First-order interactions between system and network architectures. Master's thesis, Massachusetts Institute of Technology Department of Aeronautics and Astronautics, Cambridge, Massachusetts, June 2005.
[22] D. A. Vallado and P. Crawford. Sgp4 orbit determination. In *Proceedings of AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, pages 18–21, 2008.
[23] B. Venkatamohan. Automated implementation of stateful firewalls in linux. 2011.
[24] J. Wertz and W. Larson. *Space Mission Analysis and Design*. Space Technology Library. Springer Netherlands, 1999.
[25] L. Wood, W. D. Ivancic, W. M. Eddy, D. Stewart, J. Northam, and C. Jackson. Investigating operation of the internet in orbit: Five years of collaboration around cleo. *CoRR*, abs/1204.3261, 2012.

Karim Sobh is a PhD. candidate in Computer Science at the American University in Cairo. He has achieved his bachelor and masters degree in Computer Science from the same institute. Sobh's specialization is in distributed systems and cloud computing, and his PhD. topic is cloud environments metering.

Dr. Khaled El-Ayat received his Ph.D in EE/CS from UCSB in 1977, M.A. sc. From U of T and B.Sc. from Cairo University. As a Project Manager at Intel Corp for 10 years he developed Intels first 32-bit microprocessor [80386], on display at the Smithonian Institute, Washington D.C. with Dr. El-Ayat's initials engraved on the chip. He is a Founder of Actel Corporation where he developed the first antifuse FPGA. Actel's customers have landed 60 Actel FPGAs on Mars using this technology. He is currently a Professor at the CSE Department, American University Cairo. His research interests are BCI, Embedded Systems and Computer Architecture. Dr. El-Ayat has over 40 US patents.

Dr. Fady Morcos is an Assistant Professor in the School of Science & Engineering at the American University in Cairo. He received his Ph.D. in Astrodynamics from The University of Texas at Austin, and M.Sc and B.Sc. in Aerospace Engineering from the University of Southern California. His current research interests lie in the area of orbital mechanics, trajectory design and optimization, satellite technology and satellite mission design. Dr. Morcos is a recipient of the NASA Space Act Award, for Creative and Innovative Development, and the NASA Exceptional Software Award, for designing key flight planning algorithms for NASA Johnson Space Center.

Dr. Amr El-Kadi is Professor and former Chair, the Computer Science and Engineering Department at the American University in Cairo. He was a member of the IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP) that developed the Software Engineering Code of Ethics and Professional Practices. Before joining AUC he was a consulting engineer with the Information, Technology and Facilities Department at the World Bank, Washington DC. He received his D.Sc. degree in Electrical Engineering and Computer Science from The George Washington University. Dr. El-Kadi is a Senior Member of IEEE (serving as the Middle East Representative of the IEEE Technical Committee on Operating Systems and Applications Environments), a member of ACM, and a member of Eta Kappa Nu (the US National Electrical and Computer Engineering Honor Society).