Merge/Space: A Security Testbed for Satellite Systems

M. Patrick Collins, Alefiya Hussain, J.P. Walters,
Calvin Ardi, Chris Tran, Stephen Schwab
USC Information Sciences Institute
{mcollins,hussain,jwalters,cardi,christran,schwab}@isi.edu

***Abstract*—Merge/Space (M/S) is a testbed designed to simulate multiple-agent security scenarios in satellite networks. By combining orbital data generated by a simulator such as STK with a synchronized set of images, M/S can accurately simulate bandwidth and connectivity constraints between ground stations and vehicles, enabling analyses of DoS attacks, scanning, malware infiltration and other analyses. We discuss the development of the testbed, the sample datasets included for release, and demonstrate the impact of various simulations.**

I. Introduction

Merge/Space (M/S) is a testbed which simulates multi-agent security scenarios with virtual satellite networks. Using M/S a researcher can explore attacks such as DoS, scanning, and exfiltration on moving satellites. USC-ISI developed M/S to evaluate the use of newer security paradigms in satellite networks.

Satellite systems are in transition from small constellations of custom-built designs to large collections of commodity hardware. These systems run on common operating systems, which may have concomitant vulnerabilities, for example the recently documented FreeRTOS CVE-2021-31571 and CVE-2021031572. In the ground segment, the rise of ground station as a service (GSaaS) providers such as Amazon provides increased access to antennas and the coupled risk of hostile access. Contests such as Hack-a-SAT[1] serve as proofs of concept for attacks by enthusiasts.

While this transition takes place, satellite security is subject to constraints that limit the capabilities of terrestrial information security. In particular:

- SWaP limitations which require lightweight low-power systems
- Power and communications constraints limit the resources available for patching, updates and maintenance
- Point-to-point communications obviate middlebox defenses such as firewalls, rate limiters or moving targets
- Very few system administrators are trained to spacewalk

[1]https://hackasat.com/

Workshop on Security of Space and Satellite Systems (SpaceSec) 2024
1 March 2024, San Diego, CA, USA
ISBN 979-8-9894372-1-4
https://dx.doi.org/10.14722/spacesec.2024.23074
www.ndss-symposium.org

We expect a near future security *ansatz* where satellite systems consist largely of off-the-shelf commercial hardware and concomitant vulnerabilities, while security controls and defenses built for terrestrial systems are unavailable due to these constraints. We built M/S to test new tools and faithfully simulate new defenses which exploit the unique characteristics of satellite networks.

M/S combines three technologies: Merge, SOC In A Box [3], and STK. Merge is ISI's testbed technology; a researcher can use Merge to tie together a network of physical and virtual assets into a specific configuration for testing. SOC In a Box is a virtualized SOC which provides data collection and analysis. STK is an industry standard orbital mechanics simulator. Using STK, a M/S researcher generates orbital placement, communication and other statistics, which are then fed into a network of virtual machines on the testbed. Communications are then activated and deactivated over time to simulate individual vehicles contact with other hosts in the satellite network.

In this paper, we discuss the implementation of M/S within a framework of steadily increasing simulation fidelity of attacks and discuss a path to improve fidelity in future releases. We discuss the current design of M/S and share initial results. We demonstrate how we emulate communications between moving targets and implement attacks. The technical contributions of this paper are:

- Developing a common file format, the *orbital events file*, which integrates STK and other simulators with M/S.
- The development of an exploitable satellite, the *Reasonably Exploitable Image* (REI).
- The development of a connection emulation system, using a *Channel Manager* in conjunction with the orbital events to emulate intersatellite and multi-base communications.

The remainder of this paper is structured as follows. §II describes the Merge/Space architecture, explaining the Reasonably Exploitable Image and Channel Manager in more depth and explaining the design decisions made to support this. §III demonstrates results from the testbed. §IV discusses previous and related work. Finally, §V concludes this work and discusses the path forward for the testbed.

II. Architecture

This section describes the architecture of Merge/Space. M/S adapts the Merge testbed to satellite systems by creating orbital simulation data and then using that data to feed state information (*e.g.*network connectivity) to multiple synchronized devices. This section is structured as follows: §II-A describes the general process of simulation on the testbed and the major components, §II-B discusses the ground segment composition including the red and blue networks and the SOC, §II-C discusses the space based components (*e.g.*, the REI and CM), §II-D discusses the analytics provided with this release.

A. Overview of Testbed Architecture

Figure 1 is a block diagram showing the basic elements of a Merge/Space testbed. Merge is a network testbed: its *Moa* [5] tool enables the testbed to to create networks transparently connecting multiple virtual or physical machines where each link operates within specific bandwidth and latency parameters. This capability to connect machines while simulating arbitrary network connectivity provides the means to simulate physically wire, radio and laser-based inter satellite connections, as well as simulate the propagation time differences between LEO and GEO.

M/S implements space based simulation by creating images representing ground and space components, then using orbital data vehicle placement and to consistently reconfigure communications between these components over time. Figure 1 shows that this process of testbed configurations consists of two major processes: *model generation* and *domain data generation*. These two tasks begin with a unified set of master configuration files: these files specify all the components of the simulation, their corresponding images and their orbital parameters.

During model generation, a Merge system takes a specification and creates a suitable network of images and devices. Figure 2 is an example of a specification (written as a Python program) and a visualization of the corresponding network. As this Figure shows, the resulting testbed consists of a set of *images* connected together via network links. Creating this testbed involves three steps: *realization*, *materialization* and *orchestration*. Realization is the process of identifying and reserving resources within the testbed, note in Figure 2 the use of terms such as `image` and `cores` in the node descriptions – these terms are constraints which the realization process must satisfy from its pool of virtual and physical resources. Materialization is the process of configuring the testbed based on the realization output, during this step baseline images are fetched from the image catalog and installed where necessary. After materialization, the operator can access the testbed's components via the *Infranet* control network. The final, orchestration step, is the process of configuring individual nodes via these ssh control channels in order to execute tests.

During domain data generation, the testbed uses a simulator to create timing data describing the positions, power and communications between the individual elements over time. As shown in Figure 1, the current processes converts data generated by Ansys Systems Toolkit (STK)[2]. M/S converts STK data to *orbital events file* (OEF) file format. The OEF contains testbed specific information on the satellites: positioning and a list of all vehicles in communications range. This information is indexed by vehicle and time, currently with second precision.

During orchestration, OEF data is transmitted to all components of the testbed. These components are synchronized by a single NTP server on the testbed's Infranet (a separate network used for provisioning and managing testbed nodes).

B. The Ground Segment

The ground segment of current M/S experiments consists of two subnetworks referred to as the *blue* and *red* networks. The blue network is composed of legitimate users, the red of attackers. Figure 2(ii) shows these networks via color-coding. As this figure shows, the blue networks are interconnected via a common *Security Operations Center* (SOC); Security Operations Centers are organizations which investigate security data and execute security decisions such as firewall configuration, further investigation or segmenting networks to limit attacks.

C. The Space Segment

The space segment currently consists of two systems: the *Reasonably Exploitable Image* (REI) and the *Channel Manager* (CM). These systems are synchronized using NTP, and use time-based data from the test's *OEF* to selectively activate and deactivate communications channels between each other.

Figure 3 shows the components of the REI; REI is an Ubuntu 20.04 Linux image which in its default configuration provides a mission payload, security instrumentation and exploits. As Figure 3 shows, the REI is broken into three high-level components: *agent*, *system state* and *monitoring*. In addition, the REI is connected to the network through at least two connections – one interface is connected to the Infranet, the other is connected to the experimental network, either directly or through the Channel Manager.

The agent function groups together all elements which specifically manage the REI in the simulation. This includes orbital mechanics, power consumption, and attack impact. This is all managed using a central agent application, currently implemented in python. The agent parses the orbital events files and other parameters and executes its mission appropriately. The agent also implements the REI's mission: at fixed intervals, the REI generates a data file stored on a web server for blue stations in the ground segment to download.

The system state covers elements in the target vehicle which can be affected by attacks. This includes a simulation specific filesystem (for file exfiltration, file corruption and ransomware), a set of processes, and synthetic malware. The synthetic malware is an empty process; it exists to run in the process table and have a name that can be identified by the monitoring systems as malware.

Monitoring includes a collection of common open source network monitoring and security tools. This includes OS-Query[3] to provide endpoint agent functions, Snort for IDS/IPS,

[2]https://www.ansys.com/products/missions/ansys-stk

[3]https://osquery.io

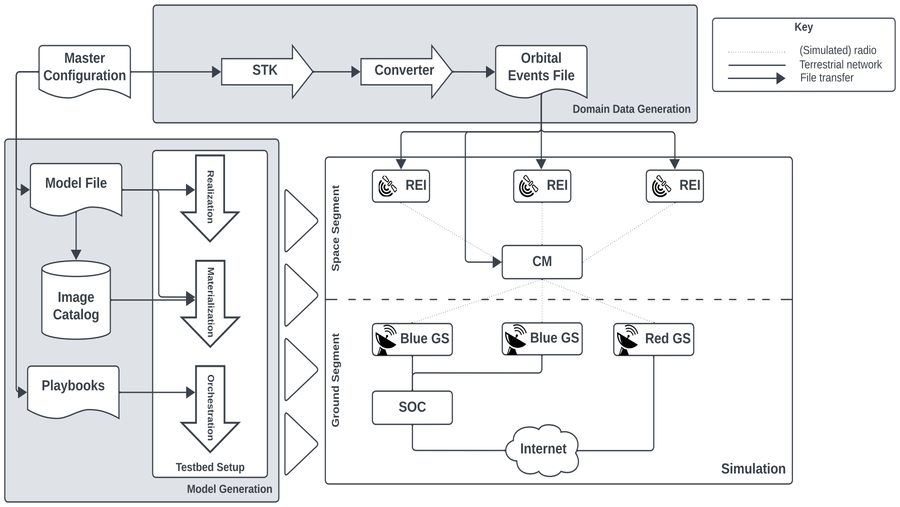


Fig. 1. M/S Block Diagram

OpenSSH configured to log the control channel, and a NetFlow sensor to generate network traffic logs.

The REI is designed to plug into the SOC-In-a-Box and provide consistent status updates. The software load is clearly excessive for a satellite, but the goal at this time was security compliance rather than deep fidelity to current satellite implementation.

The Channel Manager is a network management tool designed to reduce the complexity of simulating networking in larger constellations. The REI can logically manage connections between hosts by activating or deactivating virtual network interfaces based on the instructions in the OEF. However, larger networks leads to a large set of interfaces on very device on the network to accommodate potential connections; the CM is a transparent "orbital switch" which sits between various components in the simulation. Like the REI, the CM parses the OEF and determines which components are, at any time, within contact with each other.

The CM implements this via policy based routing. Each CM implements two routing tables, the first one contains the normal connections between all elements, the second one contains all black holes. When the OEF specifies that a particular interconnection is unavailable, the CM temporarily implements a policy which routes all connections between those two elements to the black hole.

D. Instrumentation and Analytics

Table I lists the analytic scripts developed as part of the initial release. This table lists a name of the analytic, the *vantage* of the analytic (where the relevant data for the analytic is collected) and a short description of the analytic. The analytics in Table I are not exhaustive, but provide examples of where across a testbed an experimenter can collect data and demonstrate how to collect data from multiple vantage points (as in the case of the two volume alerts).

E. Performance and Capacity

Merge/Space is built on a reference workstation using a 64Core AMD Ryzen ThreadRipper Pro and 256GB of RAM. With this configuration, we can comfortably run 80 nodes, sufficient for most historical constellations. In the current M/S implementation, vehicle state is sampled every 500 ms with approximately 25 ms precision; these figures are a product of implementing core features in Python and as we transition to more performant languages, we expect precision to increase.Currently, the simulation process is "read-only"; this limits simulations to attacks that do not directly affect vehicle placement.

III. Results

This section describes the results of initial simulations using M/S; in this section, we will demonstrate how we can simulate various forms of attacks, evaluate their impact and develop countermeasures. The remainder of this section is structured as follows: §III-A describes the data used in the simulations and which is provided as part of the public release, each following section covers a specific attack scenario. For each attack scenario, we provide the relevant indices in SPARTA [1], [2], describe the mechanism for the attack, and demonstrate the simulation in action.

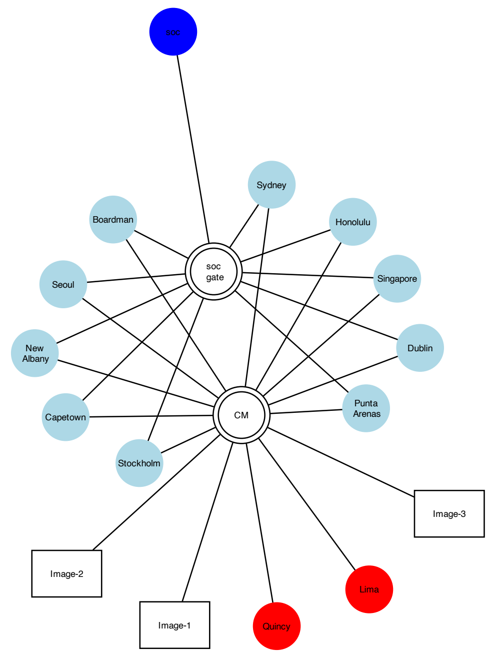
A. Data

As part of the public release of M/S, the team has provided one set of ground stations and a set of five OEF files. Figure 4 shows the ground stations used for the simulations; this set is broken into two sets – a *blue* (legitimate) set of stations consisting of 11 stations based around AWS' public stations, and a *red* (hostile) set consisting of a station in Quincy, WA,

```
from mergexp import *
# This is a topology for testing the multiplexer
net = Network('orbital')
station_names = ['seoul','singapore','stockholm','sydney',
  'honolulu','boardman','capetown','dublin',
  'lima','new_albany','punta_arenas']
stations = [net.node(i, image == '2004') for i in station_names]
station_base = '10.1'
sat_names =['landsat9','landsat8','landsat7']
sats = [net.node(i, image == '2004') for i in sat_names]
soc = net.node('soc', image == '2004')
soc_gw = net.node('soc_gw', image == '2004')
sat_base = '10.2'
scm_gd_base = '10.3'
scm_space_base = '10.4'
soc_gw_base = '10.5.1'
soc_base = '10.5.2'
scm = net.node('scm01',image=='2004')
# Build the ground segment
for index,sn in enumerate(stations):
    link = net.connect([sn,scm])
    link[sn].socket.addrs = ip4("%s.%d.1/24" % (station_base,index))
    link[scm].socket.addrs = ip4("%s.%d.1/24" % (scm_gd_base,index))
# Build the space segment
for index, satn in enumerate(sats):
    link = net.connect([satn,scm])
    link[satn].socket.addrs = ip4("%s.%d.1/24" % (sat_base,index))
    link[scm].socket.addrs = ip4("%s.%d.1/24" % (scm_space_base,index))
# Link in the SIAB

soc_gw_net = net.connect([soc_gw] + stations)
soc_gw_net[soc_gw].socket.addrs = ip4("%s.1/24" % (soc_gw_base))
for index, sn in enumerate(stations):
    soc_gw_net[sn].socket.addrs = ip4("%s.%d/24" % (soc_gw_base, index + 2))
soc_net = net.connect([soc_gw,soc])
soc_net[soc_gw].socket.addrs = ip4("%s.1/24" % soc_base)
soc_net[soc].socket.addrs = ip4("%s.2/24" % soc_base)
experiment(net)
```

(i) Network model



(ii) Visualization

Fig. 2. The CM connects the ground and space segments together

Analytic	Vantage	Notes
Ping	Application/Ground Station/Ping	Pinging hosts to verify presence.
HTTPS Transfer Log	Host/Vehicle/nginx log	Addresses mission-based file transfers; used to detect DoS and exfiltration.
Host Process snapshot	Host/Vehicle/OSQuery	Dump of all active processes for situational awareness.
Scan Alert	Network/Ground Station/NetFlow	Count number of failed connections to determine scan presence.
Bandwidth Log	Host/Vehicle/OSQuery	Total network consumption.
Host Disk Snapshot	Host/Vehicle/Script	Snapshot of mission filesystem.
Malware Alert	Host/Vehicle/OSQuery	Check for presence of known malware signature.
Volume Alert - Ground	Network/Ground Station/NetFlow	Alerts raised when data transfer exceeds a threshold; based on ground.
Volume Alert - Space	Network/Vehicle/NetFlow	Alerts raised when data transfer exceeds a threshold. Based on vehicle.

TABLE I
ANALYTICS IMPLEMENTED IN THE CURRENT MERGE/SPACE RELEASE

and another in Lima, Peru. The red stations were chosen so that the Quincy station would contend for bandwidth with multiple blue stations, and the Lima station would be able to operate without contending for bandwidth or garnering attention.

Table II summarizes the OEF data. As this table shows, we have provided two sets (regression and regression-2) for debugging; of note is that the orbital parameters in the regression dataset are completely fictional, they exist to test our ability to read files and actuate communications correctly. The other sets are all legitimate. Regression-2 consists of two vehicles, while the other sets are based on orbital data collected from public sources. The Image set is based on the LANDSAT program[4] and provides an example of a small constellation. The GPS set uses QZSS[5] as a reference set of vehicles in GEO. Finally, the IoT set is based on the Swarm SpaceBEE constellation[6]; this set was chosen for scale testing.

[4]https://landsat.gsfc.nasa.gov/
[5]https://qzss.go.jp/en/

Name	Vehicles	Orbit	Notes
Regression	2	LEO	Two vehicles with unrealistic fast orbit for testing
Regression-2	2	LEO	Two vehicles with realistic orbits
Imaging	5	LEO	Inspired by LANDSAT
GPS	4	GEO	Inspired by QZSS
IoT	50	LEO	Inspired by Swarm for scale tests

TABLE II
THE ORBITAL EVENTS FILE DATA USED IN THE SIMULATIONS

B. Contact with LEO and GEO vehicles

Figure 5 shows the results of ssh logins to simulated LEO and GEO satellites (in this case, the Imaging and GPS

[6]https://swarm.space

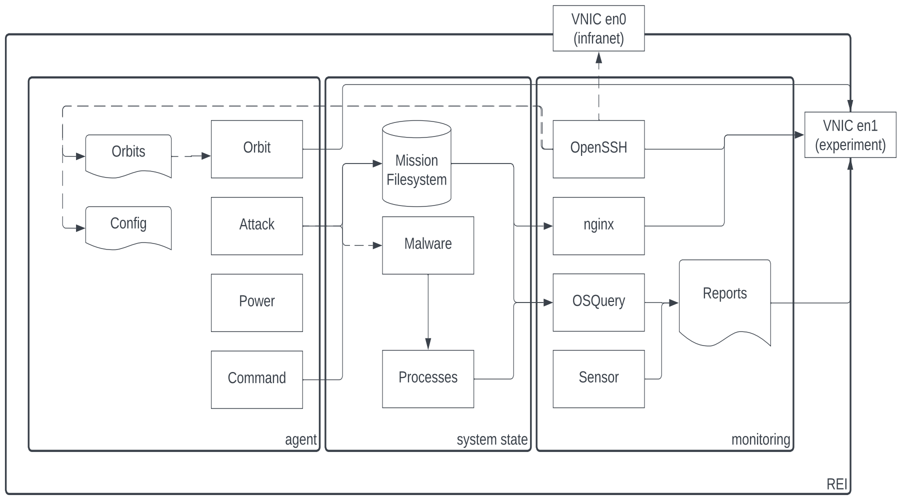


Fig. 3. REI Block Diagram

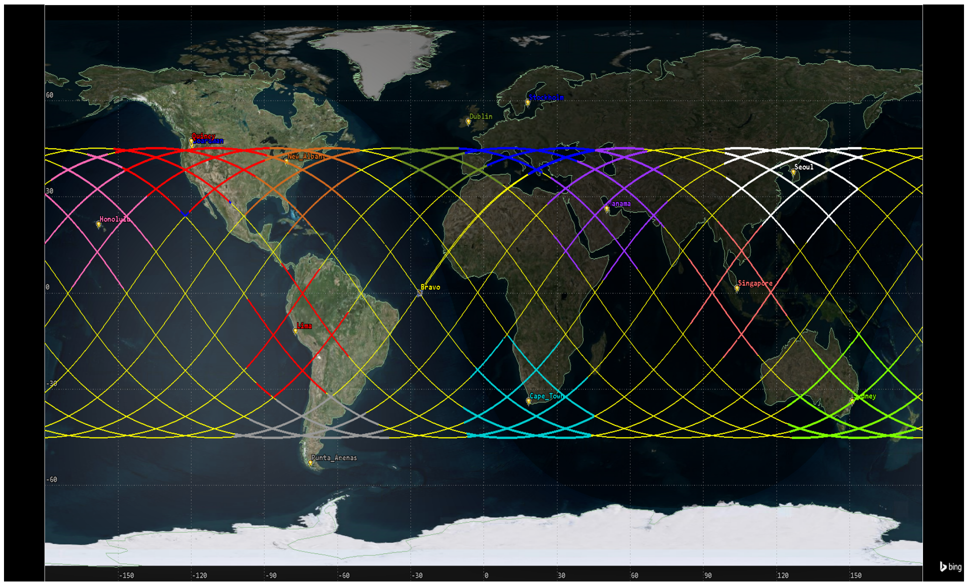


Fig. 4. Ground stations included in the test data

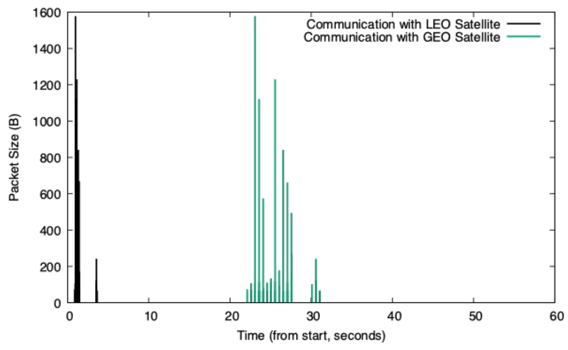


Fig. 5. M/S simulates propagation delays for LEO or GEO satellites

datasets). As this figure shows, Merge emulated different latency values for both datasets, with the GEO communications taking 220 ms to propagate a packet.

C. Reconnaissance on Moving Vehicles

a) SPARTA: IA-0006, REC-0005.04: In this scenario the attacker executes a hit-list scan, the attacker conducts an early quick scan to identify potentially vulnerable targets, then follows up some time later with more in-depth probing. The attacker is assumed to be unaware that they are attacking a satellite network and are simply scanning for vulnerable IP addresses.

The attacker's unawareness results in the phenomenon shown in Figure 6; it shows the scanning to four different

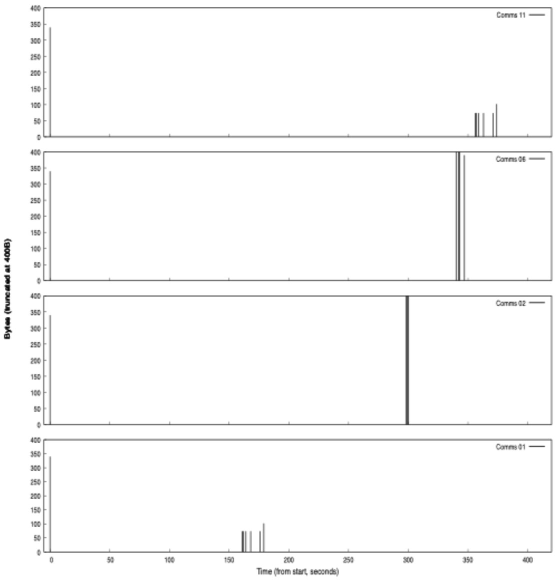


Fig. 6. Example scanning on moving vehicles

vehicles in the IoT dataset. Each plot shows packet sizes over time, with higher packet sizes indicating that the attacker was able to successfully contact the target. As this example shows, the testbed successfully simulates vehicles moving out of range, resulting in the attacker being unable to successfully re-contact those vehicles.

D. DoS

a) SPARTA: IMP-0003: The DoS scenario tested in this implementation exploits overlap between the red and blue networks. In this scenario, once the satellite is in orbit above the red station, the red station begins to rapidly extract files from the file server. The attack in this case is done by overloading the target with HTTP requests from the red station.

Figure 7 shows upstream (ground to space) and downstream (space to ground) traffic volumes for an example attack. In this figure, traffic volumes are represented as impulses in 1s intervals, with blue representing legitimate (blue station) traffic, and red representing hostile (red station) traffic. The attack is executed by running an automated fetch script

This plot shows that DoS attacks in the testbed are realistic. In this case, the aggressive polling by the red station eventually overloads the target, first choking off blue traffic, then also choking red traffic. After a short interval, the network recovers, only to again be choked by red network traffic.

IV. Previous and Related work

Existing work on satellite testbeds focuses on using components *in situ* and examining specific device vulnerabilities. Examples include Costin *et al.*'s unified testbed [4], Strohmeier *et al.*'s avionics testbed [6], and Crow *et al.*'s Triton testbed. Triton, with its combination of emulated, simulated and physical assets, most closely matches M/S's longer term goals.

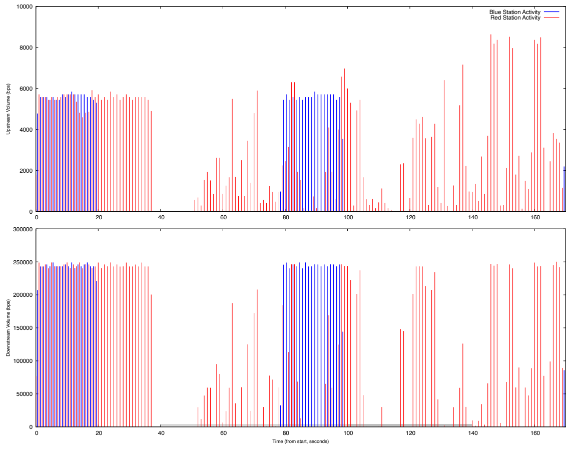


Fig. 7. DoS Traffic and Corresponding Transfer Breakdown

V. Conclusions and Future Work

In this paper we have described the core components of M/S, a prototype testbed for evaluating complex attack and defense scenarios on simulated satellite networks. We have demonstrated that we can simulate network connections between moving vehicles, and shown how to conduct attacks and observe their effects. We have integrated STK with the merge testbed and can simulate complex inter-satellite systems.

We believe that M/S is a valuable initial step towards towards developing a comprehensive testing and design exploration capability. However, the current implementation has significant validity challenges which comprise our future work. We see these challenges as falling into three major categories: improving the validity of simulations, expanding the portfolio of defensive analytics and introducing interactive domain-specific testing.

M/S's simulation work requires extensive tooling to improve the validity of the images. The REI was intended to provide a platform to run a number of off-the-shelf open source security tools which could provide analytic support without having to be implemented themselves. The next phase of this work involves improving validity by replacing the off the shelf Ubuntu installation with a bespoke image developed using a system such as FreeRTOS or RTEMS. Moving forward with proper satellite communications protocols as specified by CCSDS is another issue. Finally, validity must address the issue of power consumption.

The analytics included as part of the initial M/S package cover common attacks. We intend to expand the analytics using Aerospace's Sparta framework and the Hack-a-Sat challenges as reference points.

Another outstanding concern is to move from the non interactive OEF based simulation to an interactive one. We believe that we will accomplish this by moving away from STK to an open source a toolkit such as GMAT. We do not believe at this point that the capabilities which we will use for M/S require the full STK toolbox.

References

[1] B. Bailey. Cybersecurity protections for spacecraft: A threat based approach. Technical Report TOR-2021-01333-REV-A, Aerospace Corporation, 2021.

[2] B. Bailey. Hacking spacecraft using space attack research and tactic analysis (SPARTA). Presented at 2023 CYSAT Conference, 2023.

[3] M. Collins, A. Hussain, and S. Schwab. Towards an operations-aware experimentation methodology. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 384–393, 2022.

[4] A. Costin, H. Turtianen, S. Khandker, and T. Humalainen. Towards a unified cybersecurity testing lab for satellite, aerospace, avionics, maritime, drone (saamd) technologies and communications. In *Proceedings of the 2023 SpaceSec Workshop*, 2023.

[5] MERGE Testbed. Network emulation using MOA. https://gitlab.com/mergetb/tech/network-emulation/moa, 2023.

[6] M. Strohmeier, G. Tresoldi, L. Granger, and V. Lenders. Building an avionics laboratory for cybersecurity testing. In *Proceedings of the 15th Workshop on Cyber Security Experimentation and Test*, CSET '22, 2022.