Development of a Secure, Heterogeneous Cloud Robotics Infrastructure: Implementing a Mesh VPN and Robotic File System Security Practices

Michael Horton
Georgia Southern University
Statesboro, GA
mh09562@georgiasouthern.edu

Biswanath Samanta
Georgia Southern University
Statesboro, GA
bsamanta@georgiasouthern.edu

Christopher Reid
Georgia Southern University
Statesboro, GA
cjreid@georgiasouthern.edu

Lei Chen
Georgia Southern University
Statesboro, GA
lchen@georgiasouthern.edu

Christopher Kadlec
Georgia Southern University
Statesboro, GA
ckadlec@georgiasouthern.edu

***Abstract*—Robotics and the Internet of Things (IoT) are enveloping our society at an exponential rate due to lessening costs and better availability of hardware and software. Additionally, Cloud Robotics and Robot Operating System (ROS) can offset onboard processing power. However, strong and fundamental security practices have not been applied to fully protect these systems, partially negating the benefits of IoT. Researchers are therefore tasked with finding ways of securing communications and systems. Since security and convenience are oftentimes at odds, securing many heterogeneous components without compromising performance can be daunting. Protecting systems from attacks and ensuring that connections and instructions are from approved devices, all while maintaining the performance is imperative. This paper focuses on the development of security best practices and a mesh framework with an open-source, multipoint-to-multipoint virtual private network (VPN) that can tie Linux, Windows, IOS, and Android devices into one secure fabric, with heterogeneous mobile robotic platforms running ROSPY in a secure cloud robotics infrastructure.**

Keywords—VPN; Security; Cloud Robotics; IoT; OpenVPN; Ubuntu; ROS; TurtleBot; UFW; Heterogeneous; Robot Operating System

I. INTRODUCTION

The recent hardware and software developments in the fields of the Internet of Things (IoT) and robotics have enabled the creation of robots and devices that are more sophisticated, as well as allowing costs to decrease and availability to increase. This has proven invaluable in building a framework for IoT and showcasing what the possibilities are. However, because of the lack of standards, specifications, and architecture development, it has not fully reached its intended scale [1].

Loosely, IoT can be defined as an interconnection between all physical items, which have been given the ability to communicate. These objects become virtual entities in the cyber world and are enhanced with the sensing, processing, and self-adapting abilities to interact [2]. However, IoT is a concept that is so large and ever changing, that even experts in the field do not always agree on a universal definition [3, 4].

Enhancing this new IoT world is what James Kuffner at the 2010 coined as "Cloud Robotics" [6]. The Cloud, as defined by The National Institute of Standards and Technology (NIST), is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [7]. This definition of cloud-based computing can be expanded to cloud robotics broadly as any robot or automation system that relies on either data or code from a network to support its operation, i.e., where not all sensing, computation, and memory is integrated into a single standalone system [8]. The off-loading of these precious resources is integral in providing affordable, lightweight, and robust robotic systems. For instance, a small robot with limited computer power cannot perform complex calculations while also exercising control over its sensors and actuators. But, it could provide inputs to a networked computer and receive its instructions from that machine [9].

The basic ideas behind cloud robotics are not new. In fact, the value of networking to connect machines in manufacturing automation systems was recognized over 30 years ago [8]. Today, the field of cloud robotics is growing at an increasingly accelerated rate. The research being done today is creating opportunities like never before. In fact, in 2011, scientists in Germany introduced the idea of a new forthcoming age. The term "Industry 4.0," predicts a fourth industrial revolution that will use networking to follow the first—mechanization of production using water and steam power, the second—mass production with electric power, and

the third—use of electronics to automate production industrial revolutions [10].

Cloud Robotics and IoT are not without their challenges. It is agreed that without strong security and privacy foundations, any benefits provided by IoT will be outweighed by security breaches, attacks, and malfunctions [1, 5]. IoT security however, does not have a simple solution. It is a multidimensional subject that combines information security, network security, infrastructure security, and management security [2]. Because it is a hybrid and heterogeneous (dissimilar) network, IoT requires multi-faceted security solutions that encompasses a wide range of tasks including data confidentiality, integrity, availability and privacy, among others [12]. These reasons highlight why security was an afterthought during much of the field's development. In fact, it can be said that the 'S' in IoT stands for security. Because of this, there has recently been an upsurge of research in many related areas such as algorithms, authentications, access control, trust, and governance frameworks [2].

With the development of Cloud Robotics, a robust infrastructure such as a server farm with a reliable network is needed. The Bio-Inspired, Robotics and Intelligent Systems (BIRIS) Research Lab at Georgia Southern University (GSU) has implemented such an infrastructure. A virtualized server stack provides the necessary storage, power, and memory to handle the various research projects in progress. This system encompasses many moving parts and many students working on different projects. Even in a research setting though, security cannot be ignored in today's highly technological society and due to the multitude of components and researchers, we have to be careful in what security practices and procedures we implement, and how. Security and convenience are oftentimes at odds. We need to keep our systems safe, while enabling our researchers to operate successfully in the lab.

The rest of this paper is arranged as follows. Research questions and the investigation methodologies are presented in Section II. Section III includes the results and a comprehensive analysis of the security processes put into place. These results and details are discussed in section IV. Lastly, a results overview and plans for future work are addressed in Section V.

II. RESEARCH QUESTIONS AND METHODOLOGY

A. Introduction to Methods

This study focuses on a ROS enabled cloud robotics infrastructure and attempts to provide an answer to the following research questions.

1. How can the data located on the various robotics and the processing center be protected in the event of theft or other unauthorized physical access?
2. How can the infrastructure be secured from malicious attacks from either inside or outside of the organization?
3. How can we ensure that the robots and nodes attached to the infrastructure only accept connections and instructions from approved servers, administrative devices, and other valid nodes?

To answer these questions, a secure relationship was examined, developed, and enhanced between a private cloud-based server infrastructure and its associated IoT enabled robots. Requiring not only security at the robotics level, but also infrastructure and communication security plus IT governance principles, a combined set of security best practices for robotic file systems and communications was developed. Secondly, an open-source, multipoint-to-multipoint virtual private network (VPN) that can tie Linux, Windows, IOS, and Android devices into one secure fabric was created. This was accomplished using heterogeneous mobile robotic platforms running ROSPY; thereby creating a mesh framework where each component can securely communicate with the ROS Core, the VPN Concentrator, or even another node, without a third party being able to eavesdrop or overtake the systems. This needed to be designed and implemented with minimal latency and maximum throughput, enabling real-time processing.

The remainder of this section describes the details of the hardware, software, and setup of the various components of the BIRIS Lab used in this project.

B. BIRIS Laboratory

The BIRIS Lab at GSU consists of a server farm utilizing VMWare, multiple variations of robotic devices, and an internalized wired and wireless network. This heterogeneous infrastructure allows for the collaboration of the differing capabilities each robot brings to the system. There are currently approximately 28 virtualized servers; a few are dedicated to the operations and management of the lab, with the rest devoted to various research projects in progress. A newly upgraded ROS Core, a newly developed OpenVPN server, and a ROS receiving node are the servers used here. Each is running Ubuntu 16.04, codenamed Xenial, with the first two being encrypted, ensuring the security of the OS. ROSCore and BIRIS-VPN have a single, 1 core processor and one Gigabyte of memory, while the ROS node contains a single, 4-core CPU and 4 GB of RAM.

The network is both wired and wireless, and is sectioned off from GSU's main campus network on its own Virtual Local Area Network (VLAN). This provides internet access along with the benefit of being behind GSU's firewall and Intrusion Detection and Prevention Systems (IDS/IPS), but separate from their other systems. A Netgear router provides 5 GHz 802.11n and 802.11ac Wi-Fi access throughout the lab, which the various robots use to attach to the network. A switch provides internal VLAN's for dividing the lab's traffic. The VMWare hosts are physically protected by a locked server cabinet in a locked laboratory.

C. Robotics

There are various types of robots available for use by researchers in the lab. The current project scope includes the Kobuki TurtleBot 2, the NVidia Jetson TX2, and the Raspberry Pi3. Not utilized robots include Arlobots, Parrot AR Drone 2.0s, and ROSPY enabled Lego EV3s.

The TurtleBots are open hardware and open software, consisting of a two-wheeled platform, a Microsoft Kinect camera, and an Asus netbook running Ubuntu Linux 16.04 and ROS Kinetic [13, 14, 15]. The netbooks have 3.7 GB of RAM and a dual core, 1.1 GHz processor. These TurtleBots have multiple sensors, including infrared, gyroscopic, and collision.

The NVidia Jetson TX2 platform is a high-performance, but low-power device. The TX2 is an AI supercomputer built for complex neural networks. With a 256-core GPU, a Dual HMP processor and a Quad ARM processor, and 8GB of RAM accessed at 59.7 GB/s, the Jetson TX2 is a force or computer intelligence [16]. Connectivity includes a 1 GB Ethernet connection, 802.11ac wireless access and Bluetooth. The TX2s can be integrated into the Kobuki platform in place of the Asus netbooks for accomplishing any number of sophisticated tasks. Like the TurtleBot netbooks, the TX2s have been installed with Ubuntu 16.04 Linux and ROS Kinetic.

In 2016, Raspberry Pi replaced the Pi 2 with the third generation Pi 3. This single-board computer boasts a 1.2 GHz quad-core ARM processor with 1GB of Ram. It has built in 802.11n wireless and Bluetooth, as well as a 10/100 MB Ethernet connection [17]. Having a small form factor, using low power, and no moving parts (the Pi uses an SD card for storage), the Raspberry Pi 3 is perfect for small robotics projects. The Pi 3 involved in this project has a 32 GB Micro SD card flashed with Ubuntu Mate 16.0.4 and installed with ROS Kinetic.

D. Robot Operating System

The BIRIS lab currently runs an implementation of ROS, or Robot Operating system, by utilizing a Core-to-Node architecture. ROS is an open-source collection of libraries and tools built for creating and administering robot behavior across multiple platforms. First created in 2007, by many collaborators working together, it was made to simplify the tasks necessary to perform complex robotic operations. Willow Garage, a team of researchers dedicated to hardware and open source software for personal robotics applications, was instrumental in bringing those users together to create ROS. The ROS ecosystem is now worldwide, consisting of tens of thousands of users, working in domains ranging from tabletop hobby projects to large industrial automation systems [18]. ROS, being collaborative in nature, works well in research institutions by providing a flexible array of options and abilities.

ROS itself was not designed with security in mind since the creators viewed its use in limited environments. However, as the breadth of its use has increased, it is paramount that security practices are researched. Because of this, the bulk of the connectivity safeguards will be handled on the Ubuntu OS level. When using the defaults, a ROS Master starts and it binds to TCP port 11311, while the individual nodes use TCPROS or UDPROS to communicate back to the master. The first step in securing this connection is to implement Uncomplicated Firewall (UFW) to block connections from any machine other than the designated servers involved, including the ROS master and any approved devices. Each approved device in the infrastructure will also need UFW enabled with entries for the individual nodes, other servers, and workstations. The command 'sudo ufw default deny incoming,' rejects all but the explicitly allowed traffic and 'sudo ufw allow from *IP-Address*' allows the ROS traffic from specified machines. Finally, editing the ROS_MASTER_URI line in the ROS file bashrc specifies to the node to accept only that server as the Core. This prevents unauthorized access or misconfigurations [19].

E. OS Security Best Practices

A virtual lab was created to mimic the equipment and conditions of the production lab in order to ensure changes implemented would not adversely affect the current environment. After successful test runs of the security practices enacted, the processes and practices were moved to the production systems. Protecting the data on the various systems was accomplished by encrypting the OS upon install, encrypting the user's home folder and disabling the guest session account via a lightDM (Ubuntu Display Manager) configuration file enabling the unity greeter but setting allow-guest to false.

Preventing unauthorized access and changes to the root system required denying single user access and recovery mode access from GRUB (GNU Grand Unified Bootloader). This required editing the /etc/default/grub file and setting the GRUB_DISABLE_RECOVERY to true. As an added protection, any attempts to edit the GRUB on startup will also require a password. This was enabled by adding a user and password to the superusers group in the 00_header file of /etc/grub.d. Finally, a password was set on each GRUB menu entry. This was accomplished by navigating in a terminal window to /etc/grub.d and running the command sudo sed –I –e '/^menuentry /s/ {/ --users username {/' *. This command runs the sed stream editor. The –I trigger indicates that the file should be edited in place while the –e adds the script to the commands. The text entry that is searched is /^menuentry, with /s/ making it explicit. The line to add is specified by / --users username and finally, the * indicates that this should be run for each file in the directory [19].

F. OpenVPN

The main component of this security analysis is a multipoint-to-multipoint VPN system. This allows secure communication between the node and the core, and between nodes. The traffic inside the individual tunnels between machines cannot be viewed even by other devices on the VPN

network. The VPN concentrator chosen for this project is OpenVPN. This open-source system provides a secure tunneling connection using the SSL/TLS protocol.

OpenVPN uses an X509 Public Key Infrastructure (PKI) with server and client certificates and private keys. This is accomplished with bidirectional authentication. In other words, both the client and the server must have its own valid certificate that is trusted by the other party. After the initial connection is successful, the encryption and decryption information, along with the Hash-Based Message Authentication Code (HMAC) key data are created using OpenSSL and exchanged over the SSL/TLS connection. Both machines contribute to the HMAC Data. This traffic operates over a reliable connection, which enables the use of the User Datagram Protocol, or UDP. The encrypted packet contains the HMAC, an encrypted envelope, a 64-bit sequence number, and the data being transmitted [19, 20].

The SSL/TLS authentication session is multiplexed with the key exchange and the actual encrypted tunnel. This allows SSL/TLS to see a reliable transport layer while the packet itself sees UDP, an unreliable transport layer. This keeps the reliability and authentication separate [20, 21].

Further security options enabled include the use of TLS Authentication (TA) keys to add another layer, where an additional HMAC signature is added to all handshake packets. Finally, the default subnet was modified to provide some obscurity. Remote subnets are not enabled in the current configuration. This means that the devices will not be able to extend their communication to outside of the VLAN while the VPN is running. This can be changed, if the need arises in the future.

G. OpenVPN Configuration

The first requirement was a virtual machine to host the OpenVPN server. An Ubuntu 16.4 VM was built and named BIRIS-VPN. This was created with an encrypted operating system and an encrypted home folder. It was also given a static IP for ease of reachability by the clients. After sourcing the repository, the software package can be installed; while, unzipping the sample configurations into the working directory provides a jumping off point for configuration.

The first step is to set up the master Certificate Authority (CA) certificate and key. Installing Easy-RSA provides the necessary scripts for this task. First, we edit the vars file so that any certificates generated will have the same basic Country, State, City, etc. parameters. We source the vars file using ./vars and clear any old instances of certificates using ./clean-all. Finally, we run the command ./build-ca to create our CA certificate and key. Next, we run the command ./build-key-server server to generate the server's own certificate and key.

The VPN requires a configuration file in order to set up the tunnels on both the server and the client. Some edits of the server file include adding the tls-auth key location and key-direction. Also added were a larger cipher space using 128-CBC AES, and an increased HMAC size of SHA256. Next, after the initial connection is made, root privileges are dropped by setting the user to nobody and the group to nogroup. Lastly, the configuration file needs to be told where to look for the server certificate and key.

Setting up the client is similar. OpenVPN is first installed on the client machine. From the server, a client key and certificate can be generated by sourcing ./var again, and running the command ./build-key *clientname*. Typically, at this point, the cert and key would need to be copied over to the client, but this does add a security hole. To combat this, a script was created to create the client configuration file with the necessary certificates and keys attached. This one file can be copied over using secure copy instead, diminishing the possibility of key compromise. Lastly, the client configuration file, which has an ovpn extension, needs to be moved from the home folder on the client, to /etc/openvpn [22, 23].

Starting the OpenVPN server or client is accomplished by running the command 'sudo openvpn *server.conf*' or *client.ovpn*. Three TurtleBots were chosen for initial testing. They were installed with ROS Kinetic, the OpenVPN software and were provided a certificate and configuration file.

For management purposes, there are logs available for viewing that can show the connection stream information. In addition, a status log will show what clients are currently connected, along with their IP addresses. A management console was enabled and set up on a non-default port that can provide further granular control over connected devices.

H. OpenLDAP

The certificates and keys provided by the robots and servers during the initial connection constitute machine authentication. This is a good method to validate the robot should be accessing the system. However, this does not authenticate the user running the robot. To accomplish this, a user authentication method was implemented in the infrastructure. This second piece means that the OpenVPN configuration now uses a dual-factor authentication schema. This user login portion is tied to an OpenLDAP server recently set up in the lab.

OpenLDAP is an open-source version of the Lightweight Directory Access Protocol. This system uses a split design where the front end pulls the network information and validates it against a backend storage architecture [24]. Implementing this system as a tie-in to the OpenVPN connection mechanism now requires a user to know a valid username and password in the LDAP system. Currently, group membership is not a requirement for access. PhpLDAPadmin was installed to aid in user and group management.

OpenLDAP was integrated into the structure by using the configuration line auth-user-pass on the clients, and calling the openvpn-auth-ldap.so plugin on the server. A plugin configuration file contains the necessary information about the servers' location and administrative bindings.

I. Miscellanious

Other software was needed at various points of the project. This additional software includes Glances, which is a python-based, system monitoring application and OpenSSH, which is used for secure communication over the network. Finally, ClamAV is installed for Antivirus.

III. RESULTS AND ANALYSIS

A. ROS and OS Security Results

Uninterrupted communication is a key component of robotics; and as such, a concern in our research was the effect that running UFW would have on the performance of the machine while ROS was running tasks. System monitoring was used to carefully inspect CPU and network utilization. The CPU usage over time is detailed in Fig. 1, where a high intensity ROSTOPIC, a connection mechanism for sending ROS Messages, was running, along with UFW, while attempts were made to access the system from the attacking computer. The increase at 20:12:19 was caused by the instance of the ROSTOPIC beginning. The first connection attempt took place at approximately 20:12:34.

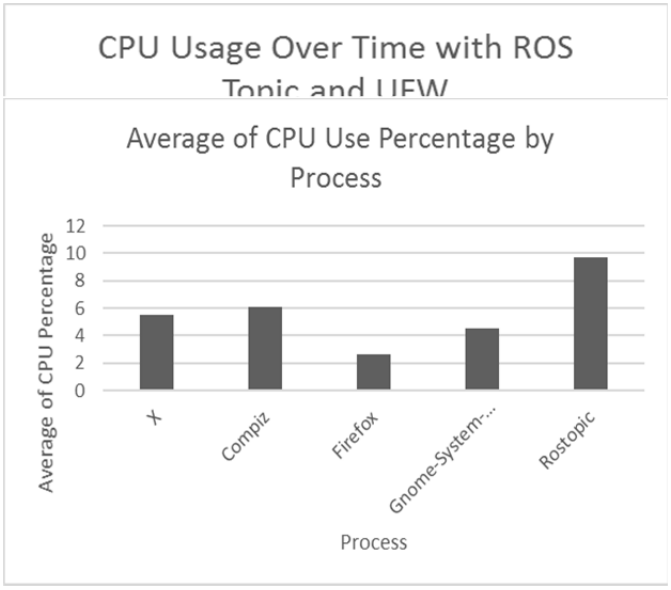

Figure 2. Average Turtlebot CPU Usage by Process.

In Fig. 2, the specific processes and their percentage of use during this test can be found. The ROSTOPIC was running at the highest percentage with 9.69%. Compiz, a GUI rendering process, measured at 6.05%. X, which includes system processes, including the UFW, averaged at 5.51%. The Gnome-System-Monitor ran at 4.56%. Lastly, an instance of Mozilla Firefox was running and averaged at 2.61% processing power. It is imperative that the security processes do not overconsume the machines resources to the detriment of the communications. Overall, the total percentage of processing power in use was 28.42%.

B. OpenVPN Results

Table 1. CPU Usage of ROS Nodes

	NO TOPICS	1 TOPIC	2 TOPICS	3 TOPICS
RECEIVER W/O VPN	3 %	14 %	14.6 %	15 %
RECEIVER W/ VPN	3.3 %	11 %	14.4 %	19.4 %
TURTLEBOT W/O VPN	7 %	22 %	16 %	19.2 %
TURTLEBOT W/ VPN	10 %	22.5 %	22 %	35.3 %

The installation and configuration of the OpenVPN server and clients was successful. There were five clients set up for the data collection phase – three TurtleBots, the ROS Core, and one receiving node.

Reiterating the communication and performance necessities, the effects of running the VPN on the devices needed to be monitored while running ROS tasks. Glances was used to carefully monitor system CPU and Memory usage, while the built-in ROS statistics tools were used to monitor latency, bandwidth, and dropped packets. A baseline communications test was run to gage system activity without the VPN running. The systems were tested with no ROS commands running first. Afterwards, a ROSTOPIC, which is a connection mechanism for sending ROS messages, was begun on one TurtleBot. The measurements were retaken, and a ROSTOPIC was started on the second and third TurtleBot, while further metrics were recorded for comparison. The ROSTOPIC used was an RGB Color Image stream from the attached Microsoft Kinect camera.

After the baselines were recorded, the ROSTOPICS were stopped and OpenVPN was started on each device. The process was repeated, recording the metrics for no topics running, and one, two, and three topics running. Table 1 details the results of the CPU benchmarks while Fig. 3 provides a visualization of the data. Likewise, Memory metrics are detailed in Table 2 with a visual representation provided in Fig. 4. Finally, Table 3 shows the average network latency of a TurtleBot with Fig. 5 providing a visualization.

Table 2. Memory Usage of ROS Nodes

	NO TOPICS	1 TOPIC	2 TOPICS	3 TOPICS
RECEIVER W/O VPN	19 %	20 %	22.1 %	28.6 %
RECEIVER W/ VPN	21.3 %	21.9 %	22.5 %	28.6 %
TURTLEBOT W/O VPN	22.3 %	27.4 %	26 %	26.1 %
TURTLEBOT W/ VPN	25 %	28.3 %	29.2 %	26.5 %

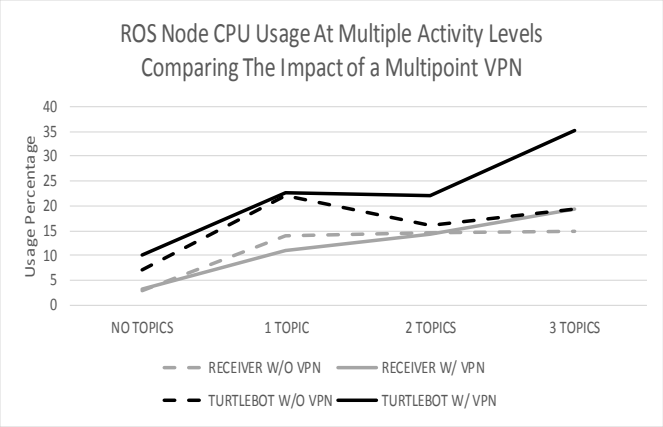

Figure 3. CPU Usage of ROS Nodes at Multiple Activity Levels.

Table 3. Average Network Latency of TurtleBot in Milliseconds

	NO TOPICS	1 TOPIC	2 TOPICS	3 TOPICS
TURTLEBOT W/O VPN	0.93	11.15	14.64	32.15
TURTLEBOT W/ VPN	2.2	11.80	17.69	28.61

IV. DISCUSSION

A. ROS and OS Security

The planning and setup of the OS and communication security practices in the virtual test environment were sufficiently laid out so that they were easily migrated to the production lab. The results of these settings in the production lab showed very low processor overhead and did not appear to impede communication between the controller, the master, and the TurtleBot. At the time of the attack, there was a 2% increase in CPU utilization; however, the initialization of the ROSTOPIC had already shown a peak usage of 12%, which was the same as when the attack occurred. There is not enough evidence from the tests performed to say conclusively that

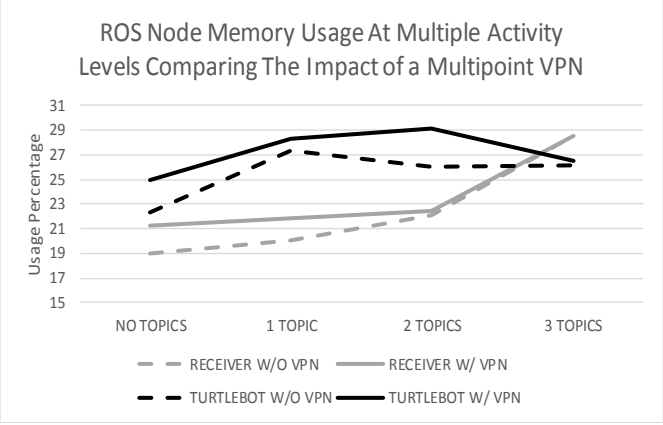

Figure 4. Memory Usage of ROS Nodes at Multiple Activity Levels.

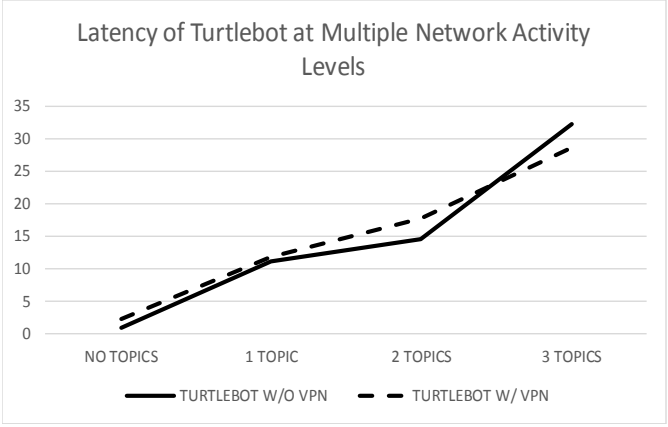

Figure 5. Average Latency of TurtleBot at Multiple Network Activity Levels.

UFW had an impact on the CPU.

B. OpenVPN

Since the TurtleBots have the exact same configuration, and the same software packages installed, metrics were only pulled from one of the systems. As expected, the metrics without any topics running for the CPU and memory are low, both with and without the VPN active. In fact, the CPU usage increased only 0.3% for the receiver node and 3% for the TurtleBot when the VPN was activated. Furthermore, the receiver's memory only increased 2.3% and the Turtlebot's 2.7%.

With all three ROSTOPICS active, the receiver's CPU increased from 3% to 15%, which is a difference of 12% without the VPN. The increase with the VPN came out to be 16.1%. There was only a 4.1% increase in CPU performance when the VPN was active. The TurtleBot had a 12.2% increase without the VPN and with, a 25.3% increase. At 13.3% less, the TurtleBot had slightly better performance without the VPN.

The memory overhead did not amount to much difference. With three topics running, the receiver averaged out to the same memory usage, while the TurtleBot only had a 0.4% higher metric with the VPN over without. In total, the receiver increased 9.6% with the VPN, compared to 7.3% without. The TurtleBot increased 3.8% with compared to 1.5% without.

The latency of the network showed an increase of 31.22 milliseconds between not having any topics running, and all three running without the VPN being active. Conversely, there was only a 26.41 millisecond increase with the VPN running. These numbers show that the VPN does not have any adverse effect on the system's performance. Additionally, there were no packets dropped in any of the tests.

Another key component of this system is that an eavesdropper should not be able to intercept the traffic during operation of the nodes. Since the VPN concentrator was a central connection point for all nodes, Wireshark was installed and enabled on the server's OpenVPN tunnel interface. A

capture was run during the test with the TurtleBots publishing a ROSTOPIC. An ICMP ping test between a node and the VPN concentrator itself was run as well. The Wireshark test showed that no traffic other than the ping was intercepted by the server, proving anonymity.

A fourth Turtlebot, two NVidia Jetson TX2s, and one Raspberry Pi3 were also added to the infrastructure after initial testing. Basic metrics were taken and proved that the software did not adversely affect system performance. The final test step on one of the TX2s was to install gnome-network manager and create a valid profile to connect the device to the OpenVPN structure. The idea behind this was to make life easier on the researchers by not having them run command line instructions.

C. *OpenVPN System Governance*

In order to provide reliable insight into the system for current and future users, documentation was created detailing each server and robot currently connected to the system, including identifying information, IP addressing, OS information, and encryption keys. Next, instructions were provided for installing the necessary application packages, creating new client certificates and keys, and loading the server and client packages on the various devices, with screenshots for clarification. Coding examples were added to explain how the system worked and errors that users may run into and corrections for them were provided.

This entire project was a lesson in risk mitigation, and having power users remain knowledgeable about the inner workings is the only way to keep up the systems effectiveness.

V. CONCLUSIONS AND FUTURE WORK

While it is an unfortunate truth that ROS does not provide any security, many other processes and practices can be implemented to secure the systems being used. The VPN implementation tested here is small in its current scale, but the project has shown that a heterogeneous network is possible and the settings can be broadened into the larger networks of today and the future. Future additions to the system include a number of platforms available in the BIRIS Lab; including Raspberry Pi 2 based ArloBots, Lego EV3s and Parrot AR Drone 2s. This will extend the heterogeneous system even further and will prove the usability of such a design.

Other expansions of this system will begin with the implementation of a secondary VPN concentrator for load balancing for increased robustness and performance. Next, the network manager configuration should be added to all available platforms that will be accessing the infrastructure to ease the use by the researchers. To increase security, a certificate authority server should be investigated to reduce the need to transfer the certificates and keys across the network. Finally, the next major testing phase includes moving a number of robots to separate VLANs in order to test the security of the system across different networks. The potential benefits for this project are vast and will greatly enhance the capability of our lab, as well as expand the scope of research and training for students.

The overall aim of this research was to provide a new way of securing these types of systems that focuses on a higher perspective than is typically researched. The practices put in place here provide an umbrella under which many platforms can operate, regardless of the specific type. The project provides a somewhat user-friendly infrastructure that researchers will be able to use confidently without impeding their own research projects and without risking their data's safety.

ACKNOWLEDGMENT

M. H. would like to thank Biswanath Samanta for his advisement, encouragement, and support during his tenure, Georgia Southern University for the use of their laboratory and Chris Reid for his knowledge of, and assistance in, the production lab environment. In addition, M. H. would like to thank Cheryl Aasheim for her "never give up" attitude. Finally, Jenna, James, and Gwen for their creativity, love, and never-ending support.

REFERENCES

[1] Y. Dong, J. H. Guo, "Research on Malicious Security Issues of Internet of Things for Mobile Internet", Applied Mechanics and Materials, Vols. 687-691, pp. 1888-1891, 2014.

[2] H. Ning, Liu, Hong, "Cyber-Physical-Social Based Security Architecture for Future Internet of Things," Advances in Internet of Things, vol. 2, pp. 1-7.

[3] K. Hodgson, "The Internet of [Security] Things," SDM: Security Distributing & Marketing, vol. 45, pp. 54-72, 2015.

[4] P. d. Leusse, P. Periorellis, T. Dimitrakos and S. K. Nair, "Self Managed Security Cell, a Security Model for the Internet of Things and Services," 2009 First International Conference on Advances in Future Internet, Athens, 2009, pp. 47-52.

[5] R. Roman, P. Najera, and J. Lopez, "Securing the internet of things," Computer, p. 51, 2011.

[6] J. Kuffner, "Cloud-enabled robots", Proc. IEEE-RAS Int. Conf. Humanoid Robot., 2010.

[7] NIST. 2010. "NIST Cloud Computing Program." Last modified July 29, 2105. Accessed November 1, 2015. http://www.nist.gov/itl/cloud/index.cfm

[8] Kehoe, Ben et al. 2015. "A Survey of Research on Cloud Robotics and Automation." IEEE Transactions On Automation Science & Engineering 12, no. 2: 398-409. Computer Source, EBSCOhost (accessed November 1, 2015).

[9] Reid, Christopher, "Networked Heterogeneous Systems in a ROS-Enabled Cloud Environment" (2015). Electronic Theses & Dissertations. 1433. http://digitalcommons.georgiasouthern.edu/etd/1433

[10] Rüßmann, Michael et al. 2015. "The Nine Pillars of Technological Advancement." In Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries. Accessed October 28, 2015. https://www.bcgperspectives.com/content/articles/engineered_products_project_business_industry_40_future_productivity_growth_manufacturing_industries/?chapter=2

[11] H. Ning and H. Liu, "Cyber-Physical-Social Based Security Architecture for Future Internet of Things," Advances in Internet of Things, Vol. 2 No. 1, pp. 1-7, 2012.

[12] S. Li, T. Tryfonas, and H. Li, "The Internet of Things: a security point of view," Internet Research, vol. 26, p. 337, 03// 2016.

[13] F. H. Nordlund, "Enabling Network-Aware Cloud Networked Robots with Robot Operating System: A machine learning-based approach," ed. Sweden, Europe: KTH, Radio Systems Laboratory (RS Lab), 2015.

[14] L. D. Riek, "Embodied Computation: An Active-Learning Approach to Mobile Robotics Education," in IEEE Transactions on Education, vol. 56, no. 1, pp. 67-72, 2013.

[15] ”TurtleBot2", Turtlebot.com, 2017. [Online]. Available: http://www.turtlebot.com/turtlebot2/. [Accessed: 27- Jun- 2017].

[16] “NVidia Jetson”, 2017. [Online]. Available: http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html. [Accessed 11 Oct. 2017].

[17] “Raspberry Pi 3 Model B”, 2017. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed 11 Oct. 2017].

[18] Ros.org. (2017). ROS.org | Powering the world's robots. [online] Available at: http://ros.org [Accessed 17 Jun. 2017].

[19] M. Horton, Lei Chen and B. Samanta, "Enhancing the security of IoT enabled robotics: Protecting TurtleBot file system and communication," 2017 International Conference on Computing, Networking and Communications (ICNC), Santa Clara, CA, 2017, pp. 662-666.

[20] Dunston, Duane. "OpenVPN: An Introduction and Interview with Founder, James Yonan - The Community's Center for Security," Linuxsecurity.com, 2017. [Online]. Available: http://www.linuxsecurity.com/content/view/117363/49/. [Accessed 10 May. 2017].

[21] "OpenVPN," En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenVPN. [Accessed 17 Jun. 2017].

[22] Ellingwood, Justin. Digitalocean.com. (2017). How to Set Up an OpenVPN Server on Ubuntu 16.04 | DigitalOcean. [Online] Available at: https://www.digitalocean.com/community/tutorials/how-to-set-up-an-openvpn-server-on-ubuntu-16-04 [Accessed 10 May. 2017].

[23] "HOWTO", Openvpn.net, 2017. [Online]. Available: https://openvpn.net/index.php/open-source/documentation/howto.html. [Accessed: 10 May 2017].

[24] “OpenLDAP” En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenLDAP. [Accessed 11 Oct. 2017].