2018-06-01

A Framework for the Performance Analysis and Tuning of Virtual Private Networks

Fridrich Shane Perez
Brigham Young University

A Framework for the Performance Analysis and Tuning of Virtual Private Networks

Fridrich Shane Perez

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dale C. Rowe, Chair
Chia-Chi Teng
Derek L. Hansen

School of Technology

Brigham Young University

ABSTRACT

A Framework for the Performance Analysis and
Tuning of Virtual Private Networks

Fridrich Shane Perez
School of Technology, BYU
Master of Science

With the rising trend of personal devices like laptops and smartphones being used in businesses and significant enterprises, the concern for preserving security arises. In addition to preserving security measures in outside devices, the network speed and performance capable by these devices need to be balanced with the security aspect to avoid slowing down virtual private network (VPN) activity. Performance tests have been done in the past to evaluate available software, hardware, and network security protocol options that will best benefit an entity according to its specific needs. With a variety of comparable frameworks available currently, it is a matter of pick and choose.

This study is dedicated to developing a unique process-testing framework for personal devices by comparing the default security encryptions of different VPN architectures to the Federal Information Processing Standards (FIPS) set of complying encryptions. VPN architectures include a vendor-supplied VPN, Palo Alto Networks, open-sourced OpenVPN application, and a Windows PPTP server to test security protocols and measure network speed through different operating platforms.

The results achieved in this research reveal the differences between the default security configurations and the encryption settings enforced by FIPS, shown through the collected averaged bandwidth between multiple network tests under those settings. The results have been given additional analysis and confidence through t-tests and standard deviation. The configurations, including difficulty in establishing, between different VPNs also contribute to discovering OpenVPN under FIPS settings to be favorable over a Palo Alto firewall using FIPS-CC mode due to higher bandwidth rate despite following the same encryption standards.

Keywords: VPN, FIPS, security protocol, encryption, network security, bandwidth, performance, framework

ACKNOWLEDGEMENTS

I would like to express my gratitude to the BYU IT faculty for their patience and assistance in guiding me through my studies in the graduate program. I want to thank Dr. Dale Rowe and the BYU Cyber Security Research Lab for providing the workspace, machines, and additional resources to use for my research, as well as the support and morale they've shared. I want to give recognition to Palo Alto Networks' sponsorship for the CSRL and allowing use of their resources for this endeavor. Lastly, I want to thank my family for their constant moral inspiration and encouragement in pursuing continuous learning.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Background & Motivation

Networks, especially those belonging to high-earning businesses and significant enterprises, are compromised often due to interference by third party attackers. Virtual Private Networks (VPNs) are high in demand among enterprises as a VPN can protect an organization's sensitive data through its use of security protocols, making it the preferred method of connecting remote data centers together. A VPN is essentially an extension of an organization's private network for remote users to join and provide the intranet to link branch offices to the enterprise network. The ability to provide a secure connection remotely is the key factor of any satisfactory VPN. To be successful, VPNs must be configured and managed correctly. For example, an organization implementing an Internet Protocol security (IPsec) VPN through Cisco devices would need to know how to resolve an out of order configuration with solutions such as enabling NAT-traversal or even testing the connection with a ping to figure out what to do next.

The devices that connect to the VPN play a major role in adhering to security configuration, such as if their prewritten operating coding would allow the VPN connections to either perform optimally or hinder it. There are two types of devices to be aware of in a work setting. The first type of device is the out-of-the-box default, in which the device's internal scripts and operating system has been preconfigured to operate solely to the standards of its designated work environment. This type of device is typically fresh out of the manufacturer's

box with only the basic applications and background processes running upon startup. The second type of device is coined as Bring Your Own Device (BYOD). BYOD is self-explanatory, in which a worker uses his or her personalized laptop or smartphone for work purposes. A BYOD may not be appropriately accustomed to the workplace's network due to different purposes and scripting. For example, a user might have social media apps running in the background of his personal computer, which would drain CPU resources and indirectly slow down productivity. As such, the differences between a BYOD and a device programmed strictly for enterprise purposes are shown through their respective network performances.

VPN design is based on the security tunneling protocol a VPN implements. Security protocol is one choice a VPN administrator needs to make. The following protocols are three common choices: IPsec, Secure Sockets Layer (SSL), and Point to Point Tunneling Protocol (PPTP).

IPsec is a security protocol designed to authenticate and encrypt IP packets during communication sessions between network devices. It ensures authentication integrity and confidentiality through two protocols, Authentication Header (AH) and Encapsulation Security Payload (ESP). The AH protocol provides data authentication and integrity through encryption algorithms like HMAC-SHA and HMAC-MD5. AH also authenticates IP packet headers and their payloads. The ESP protocol functions similarly to the AH protocol in providing data authentication, as well as confidentiality through encryption, but it only authenticates the IP datagram portion of an IP packet. IPsec operates on layer 3 of the OSI model.

SSL is a security protocol that basically provides a secure communication channel between a network machine and an Internet site host. It is recognizable as the "https" prefix in a URL on a web browser address bar. SSL uses an encryption algorithm to encrypt both the link

and the traveling data, preventing the latter to be easily read by eavesdroppers and packet sniffers as plaintext.

PPTP is a legacy VPN protocol regarded for implementing a basic point-to-point tunneling mechanism as its security measure. It is often selected as the default security protocol despite its known security issues of not having its own encryption and authentication standards. PPTP is valued because of its low overhead, making it a fast-operating protocol compared to other security protocols, whose encryption and authentication processes incur overhead and cause the network to perform slower as a result. The missing security issue is remedied with the Microsoft Point-to-Point Encryption (MPPE) protocol. However, IPsec was developed as a relatively better alternative for VPNs with having the benefits of high speed performance and sufficiently strong security settings. Nonetheless, PPTP is included within this small list of protocols for comparison purposes.

This thesis develops a framework system for measuring the network performance of different endpoint devices connecting to different VPNs within an isolated local network environment by comparing the default protocol settings each VPN uses to the encryption settings defined by the Federal Information Processing Standards (FIPS), specifically FIPS 140-2: Security Requirements For Cryptographic Modules (Caddy, 2005). FIPS are publicly announced cryptographic module standards developed by the United States federal government for use in computer systems in non-military government agencies and government contractors. FIPS computer systems to use a strict set of encryptions such as SHA.

In this research, framework is defined as a testbed environment demonstrating the processes between a client device connecting to a server under different VPN settings. The framework is validated by the following configurations and actions. The VPN architectures

would be from Palo Alto, a proprietary software, OpenVPN, an open-source application, and a Windows Server utilizing PPTP. Default settings for each VPN are compared against active FIPS settings. Network performance tests are performed through the iperf network tool, in which the interval, transfer size, and bandwidth serve as measurement units. The iperf server endpoint is set upon a Windows desktop. The comparisons are done under separate devices with Windows, Mac, and Android serving as the client endpoints. The tests are going to be conducted under these configurations to determine consistency between FIPS and non-FIPS settings and how much influence the results have on network security and speed performance. In addition, using iperf and the network router interface of the gateway router, fixed transfers of 100 MBytes are performed beforehand on each VPN setting as a solid baseline defining overhead from bytes per frame.

The results of this study provide the insight needed to confirm the direct relationship of the effect of active encryption policies on VPN network bandwidth and ultimately validate the tuning framework for future use. Depending on how the accumulated performances results differ against that of the fixed transfers, pending analyses can determine if significant deviances from the baseline are the result of system background processes or encryption overhead.

1.2 Objectives / Goals

The main objective of this study is to determine network tuning parameters by producing a framework measuring performances of different VPN types within an isolated local network environment by comparing the default protocol settings each VPN uses to FIPS-defined settings. The VPNs are individually defined by three known tunneling protocols: IPsec, SSL, and PPTP.

The first two protocols are utilized in two different infrastructures: IPsec in an enterprise VPN technology and SSL in an OpenVPN server.

From the performance analyses, network metrics such as bandwidth sent over a fixed transfer of data through the iperf network tool are going to be measured. For the sake of simplicity, it should be noted that this network performance tool is open-sourced, available to download and utilize in all involved operating systems. As for the secure protocols themselves, they are implemented according to current default practices, specifically encryption algorithms. The focus covers these protocols under different VPN infrastructures, in which case they are used as the default protocol for their respective VPNs.

Additional factors to be considered on top of testing security performance include measurement of device's CPU and memory usage, bandwidth overhead, aggregate overhead, policies set within the device, and loads to be used for performance testing. On top of configuring security settings and strength, it would also be important to keep an eye out on these factors if they demonstrate differences in network behavior during the trial run.

1.3 Problem Statement / Hypotheses

There are one research objection and one research question to be answered during this undertaking. The main objective is to a prototype framework for evaluating a VPN configuration based on the performance through the connecting personal devices. The research question inquires of the differences between a VPN using FIPS-enforced encryption standards against its default security settings. The corresponding hypothesis determines factors affecting potential differences in bandwidth between VPNs using their default settings and FIPS-supported settings to be the result of aggregating encryption overhead. These hypotheses ultimately direct the study

in developing the appropriate secure VPN framework, such as how to make full use of default options and which settings on the device require focused hardening modification.

- The development of the framework has best practice settings requiring a vendor-supplied configuration of algorithm choice and settings. The settings are to be evaluated through a series of performance tests as to determine a desired balance between network speed and security.
- For this selection, default security options provided for each VPN architecture are evaluated against a standard officially published and provided by the US federal government.
- Differing bandwidth and transfer rates between FIPS and non-FIPS settings can be traced back to factors like system CPU usage and encryption overhead.

1.4 Methodology

For this study, the computer platforms to be used for the testing were based from current popular usage. The considered platforms include Windows, Mac, Android, and Linux. Windows and Linux are the operating systems housing the featured VPNs as Palo Alto and OpenVPN prominently run on the latter, with Palo Alto based on CentOS and OpenVPN able to be hosted by Ubuntu Server, while the PPTP server can be set up on a Windows Server platform. The VPNs operate as VMs in an ESXi environment stored in one machine, which is further explained in the Methodology chapter. As for the factors involved with the VPN aspect, they have been divided and categorized into three fields, which include VPN architecture, metrics to be measured, and security protocols. See Figure 1-1 for the initial stages of the mapped methodology for the framework.

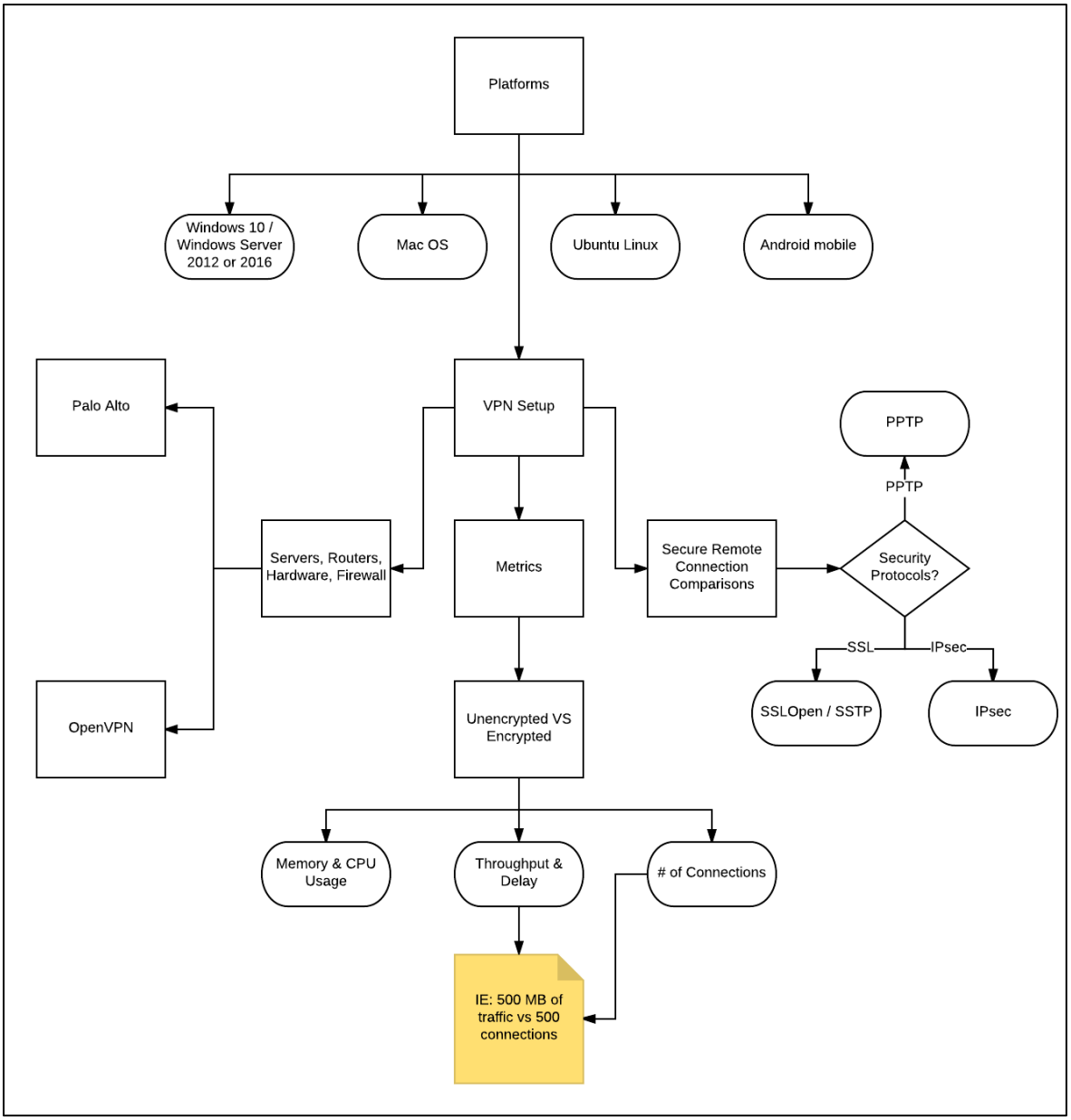

Figure 1-1: Prototype Methodology Diagram

The prototype methodology maps out the comparison of VPN infrastructures according to secure tunneling protocols such as IPsec and SSL on their effect on network speed by running performance analyses varying the settings of encryption algorithms and hardware configurations to determine useful recommendations for improvement. Parameters involved in this research

include bandwidth, CPU and memory consumption, current secure best practices, and type of platforms and operating systems used.

In addition, the methodology requires conducting thorough performance analyses for each secure VPN protocol for readily available platforms with and without active security settings. From the performance analyses, the network performance tool records transfer and bandwidth at intervals of ten seconds. As for the secure protocols themselves, they are implemented according to current practices, such as recommended encryption algorithms.

Security settings considered within scope include protocols and changeable factors such as encryption algorithm choice. Devices utilizing the network have been modified accordingly and appropriately to conform to up-to-date security and speed standards. Differences between bandwidth according to the transfer rate of packets reveal the incurred overhead as additional enforcement.

To measure network traffic with and without security settings, including FIPS-enforced settings, the iperf network tool is to be used. The tool can measure the interval, transfer, and bandwidth between devices. The results generated with it are compared with benchmarks provided by the non-VPN connection as well as a fixed transfer file test. Four test categories have been decided to be the following: one without defined security settings, one through IPsec (Palo Alto), one through SSL (OpenVPN), and one through PPTP (Windows Server).

1.5 Delimitations / Assumptions

Delimitation 1: The Palo Alto VPN is the only vendor-supplied VPN used in this research, provided by the BYU Cyber Security Research Lab (CSRL). It represents the enterprise applications factor for the framework.

Delimitation 2: The above enterprise VPN is compared with OpenVPN, the only open-sourced VPN server application to be used for this research, and with a Windows Server VM that has PPTP VPN installed.

Delimitation 3: The Windows Server 2016 VM using PTTP has been installed by scratch, using only its factory default settings. The only major configurations to be aware of regards the installation of the PPTP server, configuration of its network adapters, and the capacity to toggle FIPS settings on or off when appropriate.

Delimitation 4: One network performance test is a fixed transfer of 100 MBytes for each VPN setting as to provide an additional overhead benchmark in comparing bandwidth differences on a client.

Delimitation 5: Because the direction of this thesis involves the testing and tuning of VPN settings through personal devices, Linux VMs are not required, but can be used for additional data in need of extra clarification.

Assumption 1: Android, Mac, and Windows are the only operating systems to serve as the client side of the framework while one Windows Server OS serves as the server agent.

Assumption 2: VPN security protocols are tested by their default settings prior to experimental configuration. If default settings are not applied, the vendors' published recommended approach are to be followed.

1.6 Glossary

Bring Your Own Device (BYOD) – A term used to define a user's personal computer or smartphone being used to handle work matters. Is used interchangeably with "personal device."

Framework – A structure defining an environment where a user can freely test integral processes for analytical purposes.

Internet Protocol security (IPsec) – A protocol that authenticates and encrypts IP packets for communication over the network.

Point to Point Tunneling Protocol (PPTP) – A common VPN protocol known for its satisfactory speed performance because it lacks a defined security feature of its own.

Secure Sockets Layer (SSL) – A security cryptographic protocol that provides communications security over a network. It is commonly used in certificates for web sites, verifying their trustworthiness for communicating sensitive data through.

Virtual Private Network (VPN) – A private network enabling users to share and receive data across a public network, such as the Internet, as though they are physically connected to the network.

Federal Information Processing Standards (FIPS) – A set of public standards provided by the United States federal government that strictly define information technology standards such as encryption algorithms for use within non-military government agencies and government contractors. Validated encryption algorithms used by FIPS 140-2 include AES, Escrowed Encryption Standard, and Triple-DES symmetric keys, SHA family hashes, and asymmetric keys DSA, RSA, and ECDSA.

2 LITERATURE REVIEW

A review of VPN protocols, performance tests conducted on VPNs and VPN-related technology applications, hardware instructions for VPN usage, security approaches and developments are explored in this literature review. Topics related to encryption algorithm application, contributing to VPN security protocol strength and overhead effects on network transfer rate, are also discussed.

2.1 VPN Security Protocols

VPN security protocols define the security strength and network speed functionalities for devices communicating within the VPN area. There are protocols that define the tradeoff between network speed and security, either specializing in one over the other or attempting to achieve a balance between the two fields.

A survey took note of particular differences between the PPTP and IPsec protocols in how they provide secure communications within a VPN (Yadav, 2016). PPTP, developed by Microsoft, is a widely supported VPN method for Windows platforms due to incurring the least amount of overhead in its performance, which marks it as the fastest among VPN protocols. The lack of overhead is due to PPTP not including a defined encryption/authentication feature on its own. The remedy to PPTP's missing encryption feature is the inclusion of the Microsoft Point-to-Point Encryption (MPPE) protocol to provide the secure aspects of VPN connections.

By contrast, IPsec does not require another protocol to provide secure connections as it already possesses an encryption feature. At the cost of its own strong security and usage of encryption algorithms, IPsec suffers in incurring additional overhead and latency. Despite the performance issue, the same survey establishes IPsec as the "standard" VPN solution over PPTP with security being highly valued over speed performance.

Even with network communications preserved and protected through IPsec, the standard solution protocol has been tested with exploitation methods as to uncover any potential vulnerability. One such exploit involves injecting IPsec tunnels with an abundance of data packets in order to cause their gateways to reduce their tunnels' Path Maximum Transmission Unite (PMTU) and leave the network vulnerable to a Denial of Service (DoS) attack. This exploit is called the Packet Too Big – Packet Too Small Internet Control Message Protocol (PTB-PTS ICMP) attack. The networking world is not a stranger to DoS attacks as they are frequently used to slow down services of an organization by the hands of rival competitors or malicious attackers.

One legacy method of mitigating DoS attacks within the range of IPsec tunnels is through the Path Maximum Transmission Unite discovery (PMTUd) mechanism. There is, however, a modified PMTUd algorithm that utilizes a packetization layer protocol with an acknowledgement mechanism like Transmission Control Protocol (TCP) instead of relying on ICMP (Roca & Fall, 2014). This method prevents DoS attacks by setting up automatic time-outs in the event that such an attack is detected. However, PMTUd does suffer from the setback of ICMP packets being filtered out by routers and firewalls on the way to the sender.

In any case, IPsec is still considered for testing upon a variety of network architectures as to evaluate its overall security effectiveness, such as implementations done upon Neighborhood

Area Network (NAN) architectures to protect traffic communications (Aouini, Ben Azzouz, & Saidane, 2016). NANs make up smart grids that manage electrical appliances inside their range. Factors to consider in the IPsec-implemented NAN architecture design include dynamic data exchange and monitoring smart meters that have limited computational capacity. This study provides an evaluation framework for IPsec, and to an extension other VPN protocols such as PPTP and SSL, by establishing the physical range of the network area and setting parameters regarding network communications to measure.

2.2 VPN Applications

The application of a VPN technology for any organization is to ensure a secure network connection for its users to share data with the organization's private network. It stands to reason that the main purpose for a VPN is to prevent sensitive data, such as bank accounts or passwords for example, from falling into the wrong hands while at the same time to allow authorized users to access the private network through their devices.

An integral part of the VPN process involves optimal managing of tunnel performance in the virtual architecture (Liyanage, Ylianttila, & Gurtov, 2016). Studies by Liyanage and colleagues were done upon a similarly related network technology called Virtual Private LAN Service (VPLS), wherein participants of the VPN are connected through a multipoint Ethernet LAN connection. Tunneling mechanisms used in VPLS were described to be static, complex, and inflexible in nature. These weaknesses resulted issues in scalability, overused network resources, high delays, and high operational costs. A solution Liyanage proposed was to develop a Software Defined Network (SND)-based VPLS architecture that includes new tunneling mechanisms to remedy the vulnerabilities. The dynamic tunnel establishment mechanism and the

tunnel resumption mechanism granted additional structure to the VPLS by keeping better track of its tunnel manageability, allowing for smoother connection rates and significantly lowered resource usage rates. The idea of implementing different mechanisms for better network transition and communication is a concept to keep in mind in developing future best practices applicable for a wide variety of devices involved in the architecture.

Another field in VPNs that can be altered upon is authentication, wherein GPS information with geo-privacy protection can be implemented to enhance authentication measures (Jin, Tomoishi, & Matsuura, 2016). Users would correlate GPS with mobile devices upon thinking about which device would be best suited to use for GPS technology. Known cyber-attack methods, such as data breaching and Denial of Service (DoS), have commonly interfered with organizations' operations within their networks. Jin and colleagues utilized geo-privacy protection to enhance VPN authentication by factoring in GPS information as an additional security step on top of default VPN authentication and identification practices. Geo-privacy protection mitigates the risk of GPS information leaks by registering a valid area based on GPS information on the VPN authentication server and storing latitude and longitude coordinate bits as authentication parameters. In their research, SSL VPN was used due to SSL's design being better suited for personal user remote access compared to other formats like IPsec or PPTP. Jin's research and trial run on using geo-privacy protection was marked to be successful, wherein hit rates corresponding to latitude and longitude bits neared one-hundred percent and the overall project has been opened for future work, preferably on RADIUS server. The concept from this research inspires additional security-hardening methods not commonly found through most organizations, in which the standard use of firewalls and tunneling protocols suffice in ensuring secure network communications.

Following the lines of VPN application on mobile technology, there have been standards devised to utilize mobile devices and BYODs more frequently in the workplace based off of a survey regarding mobile VPN technologies (Alshalan, Pisharody, & Huang, 2016). Alshalan and colleagues' research on mobile security highlights the factor of technology evolution with the increase of BYODs used in the workplace. To adapt to the growing popularity of personal device usage, Alshadan's team used data from their survey, which was addressed to general IT and security professionals, to develop an explicit standard for mobile VPNs. Mobile VPN solutions such as Border Gateway Protocol (BGP) / Multi-Protocol Level Switching (MPLS)-based VPNS and mobile IPv4 IPsec VPNS were highlighted for the survey to place emphasis on creating a mobile VPN standard. Following the survey and data analysis, Alshalan and colleagues found that mobile VPN based on MIPv4 and IPsec as proposed by the IETH is the most consistent according to the desired criteria for secure mobile devices in private networks. Ultimately, the research was meant for readers to understand the technical backgrounds and open issues of mobile VPNs to inspire future work and research to develop a structured methodology and design VPN solutions that can resolve said open issues. The open issues include Software Defined Network (SDN)-enabled mobile VPN, application persistence, lightweight VPN tunnel resumption, VPN tunnel handover, detection of network disruption, and battery consumption & Network Address Translating (NAT)-ing proxies.

2.3 Commodity Hardware & Frameworks

Depending on the extent of an organization's security needs, certain aspects pertaining to a VPN should be configured accordingly. Such aspects include software, hardware, and a set of protocols for network communications to follow. However, there are also the devices, ranging

from personal computers to mobile devices, that utilize the VPN to consider if an organization wants to optimize network speed and performance for involved users.

The functionality for a network infrastructure can be highly efficient even with commodity hardware when arranged appropriately (Raumer, Gallenmuller, Emmerich, Mardian, & Carle, 2016). A study with commodity off-the-shelf (COTS) servers was performed with the intent of proving that such hardware can match the performance levels of high-end expensive networking hardware while keeping external hardware costs low and manageable. For this study, Raumer and team tested Network Interface Controller (NIC) offloading with two Intel 10 GbE NICs, which were Intel X540 and Intel 82599. These NICs were ideally within parameters for the research due to their support of IPsec encryption and authentication up to 1024 different security associations in each direction, as well as up to 128 IP addresses at the receiver side. Following a series of network performance tests to determine the true capabilities of COTS hardware, Raumer and colleagues found the results as positive. A contributing factor to the success of proving commodity hardware matching high-end expensive equipment is the open source device driver the Intel NICs used, which was based from the Data Plane Development Kit (DPDK) and MoonGen traffic packet generators. In additional, while the idea of IPsec capabilities working on NICs is nothing unfamiliar in the networking world, this research is marked as the first case study to actively engage into testing hardware performances instead of standard tunneling protocols with network performance parameters.

In addition to hands-on testing with commodity hardware selection, frameworks offering faster alternatives to compensate for hardware limitations, such as packet rates given by 10 Gbit Ethernet, were examined for the potential gain of reducing overhead and CPU-induced bottleneck (Gallenmüller, Emmerich, Wohlfart, Raumer, & Carle, 2015). Existing software

frameworks, namely netmap, Intel DPDK, and PF_RING ZC, were surveyed on Linux networking for high-performance packet IO under the limitations of network bandwidth, CPU cycle performance, PCI express bandwidth, and maximum transfer rate determined by Ethernet standards. Like the previous study on commodity hardware, this research utilizes two 10 Gbit NICs for packet handling. The three packet IO frameworks require modified drivers and share the same techniques in packet performance, which include bypassing the default network stack, relying on polling to receive packets instead of interrupts, and preallocating packet buffers at the start of an application with no further deallocation and allocation or memory during execution of an application.

Once the theoretical aspects of packet IO forwarding were established from existing frameworks, measurement setup was focused on integral factors such as CPU cycles per packet and throughput to compare the framework performances. This research discovered additional factors that affect packet IO performance, which include cache influence, batch size influence, and latency. Ultimately, large batch sizes were shown to increase performance and latency in addition to the trade-off between queue sizes, concluding in DPDK and PF_RING's superiority over netmap in terms of these parameters. All things considered, it is a matter of proper application of a framework through modified system interfaces that would allow high performance rates while keeping in mind network interface limitation to avoid crashing it.

2.4 Performance Testing

To determine the effectiveness of software, hardware, and protocol implementation in VPNs and general network architecture, statistics regarding network performance and traffic need to be obtained for comparisons. Depending on the research question and problem

statements supporting the inquiry, metrics and parameters can generally vary. However, most network performance tests require the analysis of collected bandwidth and packet transfer rates to verify or disprove speed between different software and hardware.

A study was done regarding the interoperability concerning transition mechanisms between IPv4 and IPv6 networks (Narayan, Ishrar, Kumar, Gupta, & Khan, 2016). Considering how IPv4 has been used for over thirty-five years, making the change into IPv6 would not be a quick and easy task. End to end network performance tests were conducted to determine the most effective method of utilizing transition mechanisms developed by the Internet Engineering Task Force (IETF) to allow an IPv4 network transition into the more advanced IPv6. Out of the three discussed transition mechanisms, which are dual-stack, tunneling, and translation, the tunneling mechanism was placed in focus for the testing. The performances were tested under Windows 7 and Windows Server 2012, as well as with and without PPTP and IPsec protocols active. It was found that the 6to4 transition protocol is the faster and more reliable mechanism compared to 4to6 due to having significantly lower delay among consistently reliable results. In addition, 6to4 with IPsec produced the highest amount of TCP DNS throughput compared to the other three combinations involving 4to6 and PPTP. Future work based from this study may include testing on different platforms, such as Mac and even the Android mobile platform, to determine more efficient systems combining speed and security.

There is also the basis of understanding the performance differences in operating with and without an active security protocol like IPsec, specifically on processing delays on automated nodes for this study (Hirschler & Sauter, 2016). The research focusing on these parameters tested within resource-limited devices using Intel and ARM-based architectures on System on a Chip (SoC) hardware. The driving problem statement for this research was to

provide feasible security features for automated smart grid applications as IPsec was originally designed for IT environment, not automation networks. The transmission and receiving delays were measured upon seven series, starting from normal IPv6 transmission before combining it with two encrypting algorithms used in IPsec, which are Authentication Header (AH) and Encapsulating Security Payload (ESP). On top of testing without IPsec, six different combinations were made from the encryption algorithms through IPsec's tunneling and transport modes by testing both modes with AH, then ESP, and finally both combined. Collected test data proved additional security features, namely the combined encryption algorithms, incurred more overhead compared to plain IPv6 transmission. In addition, IPsec tunnel mode with encryption algorithms incurs higher delays than transport mode under the same corresponding settings. As for hardware, Intel has shorter transmission and receive delay ranges with the implementations of AH and ESP compared to plain IPv6 than ARM.

There is also a focus concerning the performance of IPsec, determining if hardware implementations do affect speed (Rao, Newe, Grout, & Mathur, 2016). One study holds a claim that hardware implementations on security algorithms provide high-speed and real-time performance for applications like data integrity and confidentiality. Considering that IPsec is computationally intensive to secure the transfer of data, this issue would warrant attention involving the improving of its speed. An attempted hardware implementation involved a field programmable gate array (FPGA) capable of dedicated operations that can provide more consistent and higher performance statistics, at least when compared to software implementations, with a SHA-3 encryption algorithm. The idea behind utilizing FPGA is that it is considered as the best leading representation hardware devices of the modern era, shown with significantly higher throughput compared to previous attempts.

This research is notable for being the first published work showing hardware implementation on IPsec without any soft-core processor involvement while permitting a major SHA-3 contribution in the sense of using the algorithm to handle IPv4 datagrams through IPsec's transport and tunnel modes.

Additional performance testing on IPsec, in an attempt to compare its throughput and round-trip time (RTT) with that of SSL, has been done on a Windows 7 platform and 802.11n WLAN (Kolahi, Cao, & Chen, 2013). Additional details in this performance testing include comparing the Windows 7 and WLAN to open system and using Triple Data Encryption Standard – Secure Hash Algorithm (3DES-SHA), and Advanced Encryption Standard – Secure Hash Algorithm (AES-SHA). This performance testing show that IPsec VPN had the significantly better throughput and RTT performance than SSL. While it is true that using strong security settings lowers network performance and increases delay, IPsec had a maximum decrease of 60.28% in TCP throughput compared to open system while SSL had a maximum decrease up to 97.03%. In the same comparison to open system, bandwidth with IPsec dropped 50% while bandwidth with SSL dropped 96%. As such, network settings with IPsec won't suffer much delay or low throughput than with SSL, making IPsec the better choice in this scenario. This research has opened itself up to future experimentation with other VPN technologies, such as PPTP and L2TP, as well as the opportunity to utilize other operating systems like Linux in this case study.

Aside from individually testing software, hardware, and protocol options for private networks, performance testing has allowed the capacity to appropriately modify enterprise and campus-sized networks with Virtual Local Area Networks (VLANs) to control the broadcast traffic and lessen the chance for congestions to occur (Ashraf & Yousaf, 2016). This can address

an issue of inter-VLAN routing, wherein a host from one VLAN communicates to a host of another VLAN through a forwarding layer-3 device. The issue is that security would be at a high risk with the enterprise network segments being geographically dispersed to the extent that outside unauthorized parties could take advantage of the congestions between segments and slip into the network undetected. Security perspective of VLANs and inter-VLAN routing was investigated through IPv6 protocol and IPsec tunneling, ultimately constructing a secure architecture from evaluating these security measures. The IPsec virtual tunnel interface is created through the "interface tunnel int-number" command while comparing IPv6 with IPv4 performance. The results of the traffic simulations are collected as round-trip delay (RTD) and are compared under four situations: using a layer-3 switch with no security, using different switches with no security, using different switches through IPsec transport mode, and using different switches through IPsec tunneling mode. Running IPv4 with no security had consistently low delay while IPv6 under IPsec tunneling mode achieved low delay almost reaching the former; both results utilized different switches. For the sake of overhead efficiency and security, it is concluded that IPv6 routing with IPsec tunnel mode would be the ideal method to implement in an IPsec VPN.

2.5 Encryption Algorithms

The functionality of a security protocol relies on its capacity to utilize encryption algorithms to encode data to the point that intruding outside parties cannot decipher the communication. There is a tradeoff between speed and security concerning encryption algorithms as the more computationally intense they operate in securing date, the slower the system communicates.

For the most part, security is highly favored regardless of intensity or complexity, especially for a system that is willing to entertain the idea of a multi-phase encryption technique in place of typical algorithms like Advanced Encryption Standard (AES) and Blowfish on top of traditional VPN security settings (Singh & Gupta, 2016). The purpose for increased complexity aside from strengthened security is to prevent data tampering in case a compromised link occurs in the network.

AES has been performance tested on a 2010 Intel Core processor due to previous research finding the correlation between the AES cipher behavior and processor core scaling (Uskov, Byerly, & Heinemann, 2016). Hardware acceleration and a set of new instructions (NI) written for AES that can execute under significantly fewer clock cycles than a software solution provided the test settings needed to compare with AES under default settings in the same Intel architecture. This testing of AES-NI involves running AES with RMM ready-to-be-streamed files of sizes 100 MB, 200 MB, 500 MB, 1000 MB, and 2000 MB with the AES-NI instructions enabled and disabled. Lastly, the AES cipher operations are divided into five different modes, which are CTR, ECB, CBC, CFB, and OFB. These AES modes are what would define the final results of the performance analysis, determining which one would work the most efficiently or had the best median. Following the testing on the AES-NI hardware acceleration, the research concludes that the AES-ECB mode had the best median performance out of the five AES modes with AES-NI enabled. Also, with AES-NI disabled but still running with the same Intel hardware, AES-CTR gained the best median performance, but was still beaten by ECB mode. However, the findings do not recommend using ECB mode for secure VPN networks as it cannot completely hide encrypted data patterns, and as such, recommend CTB mode instead as CTB mode had the second-best performance in AES-NI enabled mode.

There is also the concept of implementing encryption and decryption algorithms through graphics processing unit (GPU) technology (Heinemann, Chaduvu, Byerly, & Uskov, 2016). By correlation, the use of OpenCL and CUDA software platforms, the latter specifically integrating with NVIDIA GPU technology, would entertain the idea of additional effects software implementations place upon the efficiency of encryption and decryption. This concept would then extend to how the changes in algorithm behavior could be affecting secure network traffic performance, such as if the enhanced complexity is slowing down the communication rate.

As for FIPS-compliant algorithms, a case study was done in which a Triple Data Encryption Algorithm (TDEA) was to be implemented in a FIPS 140-2 defined module (Barker & Barker, 2012). Modes of operation for TDEA are specified by SP 800-38: Recommendation for Block Cipher Modes of Operation. To qualify for compliance, TDEA was designed from the preceding DEA specified in FIPS 46, effective July 1977 for the data protection of federal agencies prior to withdrawal. The DEA cryptographic engine protected blocks of data consisting of sixty-four bits under a sixty-four-bit key, the functions now implemented in TDEA. The applications for TDEA were directed for environments requiring strong cryptographic protection, but the testing for TDEA was formatted in a way that its implementations would depend on the needs of the environment in question.

3 METHODOLOGY

This thesis focuses on one research objective and one research question:

Research Objective 1 (RO-1): Develop and test a framework based on best practice settings requiring a vendor-supplied configuration of algorithm choice and settings and their effects on overall network performance.

Research Question 2 (RQ-2): What are the differences between a VPN using FIPS-validated encryption algorithms compared to its vendor-provided default security settings?

Research Hypothesis 2 (RH-2): Differences between FIPS and non-FIPS settings are found in bandwidth and bit transfer attributed to encryption overhead.

The answers and processes for the above objectives are detailed by the following VPN study framework, metric assessment, secure best practices, VPN configurations, and network performance analysis methodology in this section.

3.1 RO-1: Framework Development and Testing

The development for the framework began with the purpose of assessing the viability of VPN security protocols and determining a correlation between their effect on network performance. The practicability of VPN protocols using their default settings and encryption algorithms would also be tested against FIPS-enforced encryptions. The concept behind this framework is to determine the ideal security settings derived from a vendor-supplied

configuration and apply appropriate changes either to the network setup or to the personal devices using the VPN to allow optimal network speed and security performance. The intent behind starting with default settings for VPN and personal device configurations is to build the framework from scratch and branch off into a different direction than previous network performance tests were used for. As such, this framework development project is simply codenamed "FAVE," an acronym for FIPS Application VPN Evaluation.

Related existing frameworks on the topic were found, most of which are composed of VPN architectures that performed comparisons on factors for hardware CPU and memory usage, the strength between encryption algorithms, and selected platforms as described in the literature review chapter. These factors were tested within a selected VPN infrastructure at a time. Despite these findings, there was no specific framework that focused on testing the private network reachability between different VPN security settings under the same platforms, or at least under the VPN types and client devices used for this research.

FAVE has been developed from scratch due to a lack of sufficient existing frameworks matching the direction of this research. The development consists of two parts, setting up the VPN server and running network performance tests on a selected device within the VPN while under a security protocol. The configurations for each VPN are bound to vary in time and in setup methodology, which include either a few clicks to enable certain options or opening a Terminal or Command Prompt interface to input needed lines to install or activate specific processes. And as mentioned previously, the resulting network statistics are collected through the iperf network tool, which is executed via Terminal or Command Prompt on both the client and the server sides. See Figure 3-1 for an illustrated prototype of the framework's processes in VPN performance testing.

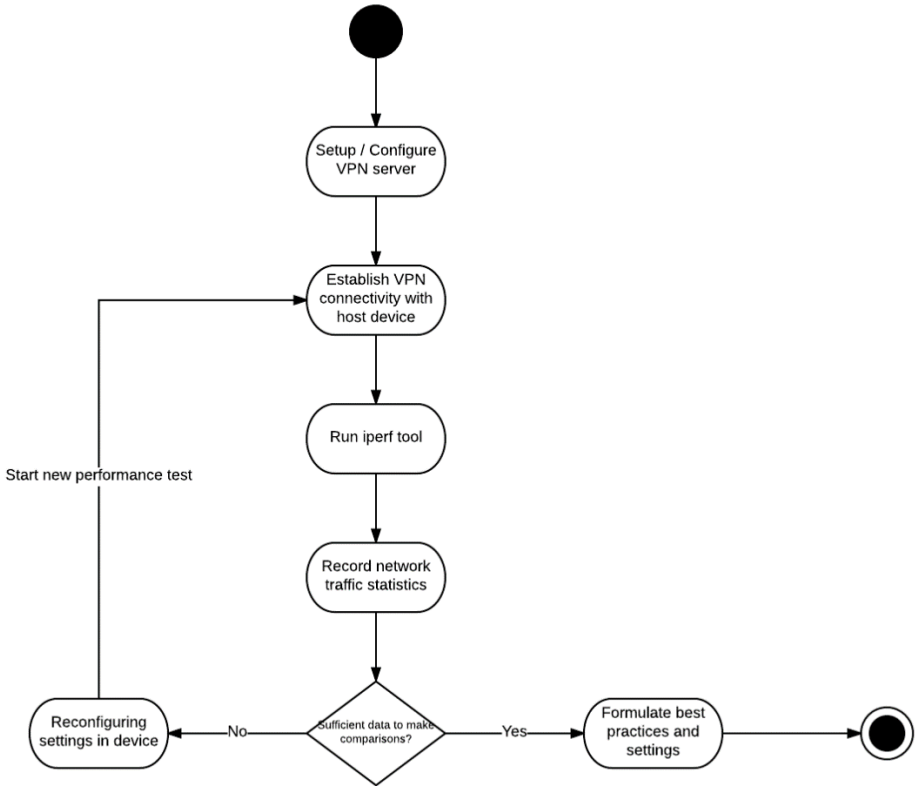

Figure 3-1: Flowchart of Framework Prototype

3.1.1 Network Performance Testing

The general purpose for network performance tests is to evaluate traffic behavior and determine factors that influence its transfer rate. For the typical user, he or she wouldn't concern themselves with the deeper and intricate details of network architecture that affect speed and security if the user is able to connect to the desired destination address without issue.

The general process of performing a network performance test is simple and self-explanatory. There exist tools, ranging from open-sourced to enterprise-created, that can simulate network communication from one end to another by sending data packets to a destination IP address and test the state of said communication. The receiving end would then respond with an acknowledgement (ACK) signal if communication has been successfully established or a

negative acknowledgement (NACK) if an error occurred that prevents the packet from being received. There is a distinct difference between a packet being dropped in its path and being rejected by the endpoint due to firewall settings, for example. From this, the network tool would measure the time it took for the test packet to reach the destination and the rate that the packet hops in between devices to reach the destination.

The framework prototype started with the idea of running performance analyses varying the settings of encryption algorithms for each VPN under the same devices. There are four main test scenarios: one without defined security settings, one through IPsec from Palo Alto equipment, one through SSL from OpenVPN, and one through PPTP from a Windows Server VM. The VPN scenarios are to undergo additional testing between using predefined encryption settings and using FIPS-enforced encryption settings, reaching a total of seven tests for each client endpoint. These factors have been incorporated into the framework's final development.

3.1.2 Network Measuring Tools

To test the network traffic, a tool is needed to measure and simulate the communicating data stream from one end to the other. There were a variety of known open-sourced network performance tools to select from that can get the job done as options included iperf, netperf, nuttcp, and netpipe. Regarding scope and the capacity to measure network performance metrics, these tools fit the basic qualifications for this research. Each one is capable of simulating data streams through client and server agents, provided that both end points have the tool installed.

For the performance analyses, it was decided that the network metrics would be recorded through the iperf v2 tool. Iperf is a free and open-sourced network performance tool that is easy to learn after a few trials of experimentation. It can measure the interval, transfer, and bandwidth

between devices. However, there is a distinct difference between iperf v2 and the rewritten iperf3 as the latter is not backwards compatible. The reason that the lower version of iperf was selected in consideration for the Android platform involved in this research. The iperf v2 tool was found to be the only compatible and available network performance measuring tool for said platform.

As for netperf and netpipe, while they were found to be compliant with Linux and most UNIX-based systems, there were no feasible applications of them for Windows and Mac. While this restriction is not necessarily a bad thing, the restriction kept in mind that the tool needed operate well in Windows and Mac. The additional requirement is that the tool is available for Android, in which there exists an iperf app as mentioned above.

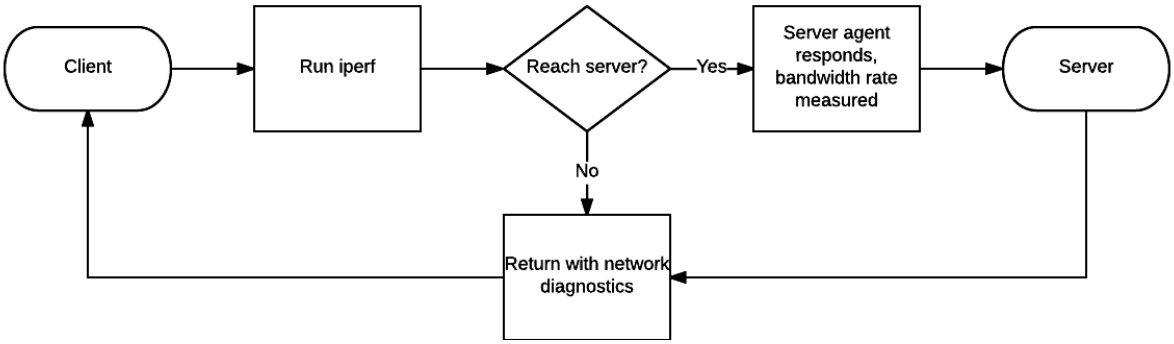

Figure 3-2: Iperf Client and Server Interaction

In addition to selecting a pre-existing network performance measurement tool rather than creating a tool with similar functionality, iperf was worked with mostly following previous network-related projects, adding in the convenience of experience it provided. It was found that developing methods to be best based from experience while learning about new tools and technologies along the way in developing an effective methodology. The iperf tool is relatively easy to utilize following download, simply by opening a Command Prompt or Terminal screen

and executing the iperf command with the -s flag to open the iperf server agent on one device while the -c flag uses the second device as an iperf client with the IP address the server client is on accompanying the flag. Iperf results can also be outputted into a txt file by adding "> [filename].txt" as part of the iperf command; it works for both the client and server commands. Additional notes regarding iperf usage that should be known is that for this research, the iperf client connects to the server via TCP port 5001 by default for all the involved platforms.

3.1.3 Framework Arrangement

The network map for this research was designed to operate separately and unaffected by other network activity and traffic in the shared lab environment. The isolation would allow a deeper study of the effects of VPN tunneling on network data packet transfer between server and client. The main access point router in FAVE would only be hardwired to the Internet only in cases that require outside resources, such as downloading tools and software patches. Otherwise, the only machine the router would be wired is the ESXi environment containing the VPN VMs while the client endpoints are permitted to connect wirelessly. The desktop server endpoint, however, would be designated on a separate subnet to prevent the clients from circumventing the VPN tunnels to connect straight to it. A fixed physical distance, d, is also established between the client device and the wireless router to ensure that all clients use the same parameters during the testing. See Figure 3-3 for visualization of the developed framework, including assigned IP addresses for machines involved and the subnet range listed on the top left.

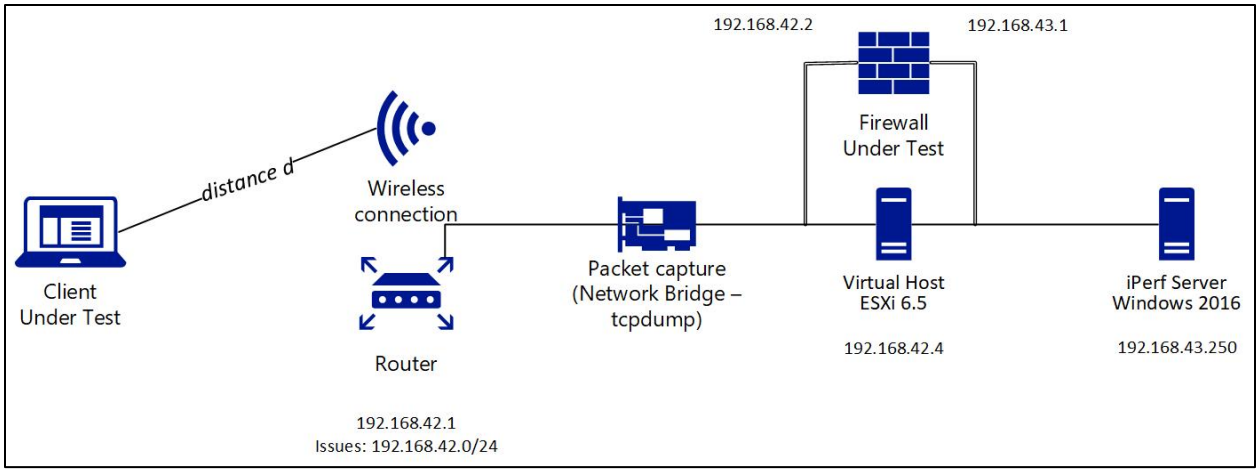

Figure 3-3: FAVE Framework Detailing Basic Relations Between Endpoints and VPNs

The reason that the VPNs are to share the same local IP address is for ease of swapping between VMs on top of consistency. Sharing the same IP address also maintains the integrity of FAVE's subnet assignments as altering a VM's network interface tends to affect its behavior in connecting with the rest of the network.

3.2 RQ-2: Determining Differences

To identify the probable difference in configuration between a vendor-supplied security setting and a federal standard of validated encryption algorithms for a VPN infrastructure, measurement parameters were decided to define obtainable variances. Measurement parameters involved in this research include bandwidth, transfer rate, current secure best practices, and type of platforms used.

All participating devices, personal or otherwise, are to adhere to the same ethical and behavioral standards. The Palo Alto firewall and OpenVPN VM setup would be built according to the basic and necessary functionalities as recorded on official documentations. These VMs, along with the Windows Server VM running a PPTP VPN, would be stored into a VMware ESXi

hypervisor. As previously illustrated in Figure 3-3, the ESXi hypervisor manages the VMs, permitting VMs to be active or suspended depending on which VPN requires testing during the experimentation period.

However, to ensure the flow of network traffic pushes through the designated active VPN's tunnel, the VPN VMs have been configured to utilize the same static IP address and host the default gateway for the desktop server in a spare virtual ethernet interface. Otherwise, due to the arrangement of the isolated network, the iperf testing would circumvent the VPN tunneling to reach the corresponding endpoint in the desktop server.

3.2.1 Network Traffic Monitoring

There are plenty of open-sourced network measuring tools and graphical user interfaces (GUIs) available to select and perform the data gathering. However, as discussed previously, the use of the iperf tool is enough to measure the likes of bandwidth and transfer rate for each VPN connection the platforms participate in.

Iperf is open-sourced and versatile in its functionality, from allowing a finite number of tested intervals to be performed consecutively to sending a file of a fixed number of bytes from client to server. The server agent is active by executing the tool with the -s flag while the client side is voluntarily activated by -c followed by the IP address where the active server is located. The "-c <server address>" argument alone measures the bandwidth and transfer rate for an interval of ten seconds by default.

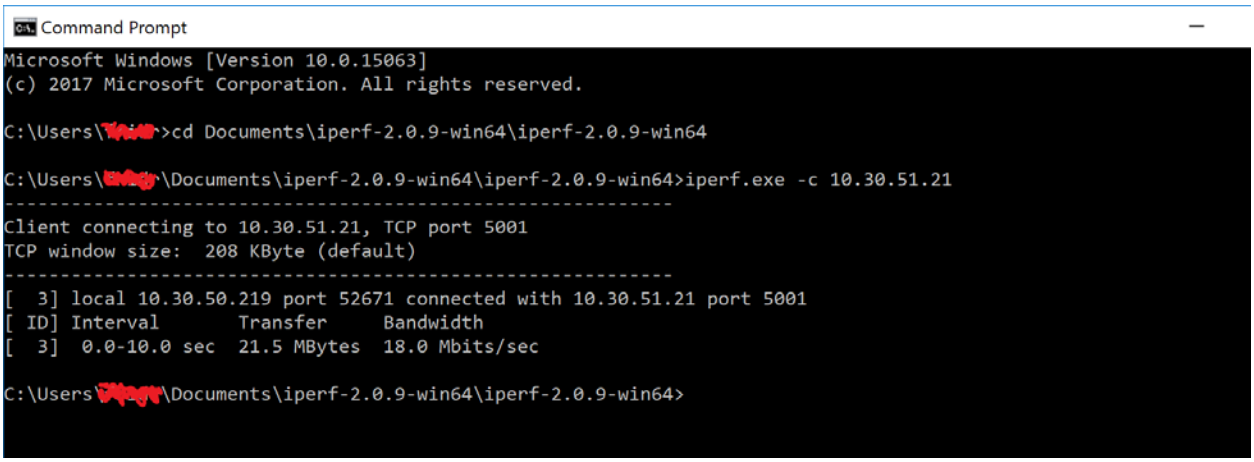

Figure 3-4: Example of Windows Iperf Client Agent Connected to Active Server Agent

Specifically, for this research, iperf tests are set to a minimum of twenty consecutive tests with each interval measuring ten seconds each. The transfer rates from all intervals are added up while the twenty bandwidth results are averaged. This iperf command is performed with arguments "-t 200 -i 10" on the client side, in which the t flag defines the total length of time while the i flag defines the interval length within the time total. To preserve the results from the performance tests, iperf can export and save them if the ">" syntax is used at the end of the command followed by the location and name of the file to save as, preferably into a simple txt file.

To validate the collected data, fixed transfer tests of one-hundred megabytes are performed at least once for each VPN setting with one client device to determine byte overhead. The Windows 10 client would be used for this case. The fixed transfer test is done with the "-n 105000000" argument. The purpose for this iperf command is to help monitor the byte rate traveling through VPN tunnels.

The actual packet monitoring would be performed through the interface of a ProLiant DL165 G6 Basic SATA containing the SecurityOnion software; both the router and the ESXi

box are hardwired to it. If setting up SecurityOnion for the first time, follow the production deployment instructions available on the Security Onion wiki, starting with the standard "sudo apt-get update" before installing the meta-package itself with "sudo apt-get -y install securityonion-all syslog-ng-core". The integral part to be able to monitor network traffic would be configuring the /etc/network/interfaces file to build a bridge to listen from, as well as define the two ethernets in use. See the Appendix B section for the exact configuration.

The packet monitoring and dumping is executed by tcpdump with the captured packets saved into pcap files. The command used to measure most network streams is "tcpdump -i br0 -w file.pcap", wherein br0 is the bridge between ethernet connections, the ASUS Wireless Router and the ESXi box, to listen traffic in.

Under a packet analysis software application, such as Wireshark, additional assurance that the packet flow is forced over the VPN is guaranteed upon examining the pcap file's data conversations and checking the active ports. On Wireshark, load the pcap file, go to Statistics, and select Conversations to view the port communications and number of bytes transferred. In this case, TCP port 443 is highly sought after while the appearance of TCP port 5001, the default port used by the iperf server agent to listen upon, would indicate the packets circumventing the VPN tunnel and traveling to the destination directly through the isolated network instead.

Following the data gathering, appropriate comparisons would be drawn between measurements respective to their fields before drawing conclusions on the best course of actions of finding the optimal balance between network speed and security for personal devices connecting to a VPN through hardening procedures and best practices.

3.2.2 VPN Infrastructure Settings

The three different VPN infrastructures presented in this research are each running separate security protocols, varying by encryption use and traffic regulation. With each VPN defined differently by protocol, the tuning factors between their default security settings and FIPS-applied encryptions should be carefully defined for clear results. Evaluating the infrastructures individually before performing network tests would provide the insight, understanding, and direction needed on developing a balanced application between network speed and security from available configurations.

Palo Alto Networks uses GlobalProtect Gateway as its network security platform for devices connected to its VPN while its firewall interface operates it from a distance. Its default security protocol is IPsec. GlobalProtect uses IPsec Crypto Profiles to specify authentication and encryption algorithms. The default algorithms for encryption from most-to-least secure are AES-256-GCM, AES-128-GCM, and AES-128-CBC. The GlobalProtect agent is applicable for Windows, Mac, Linux, and Android platforms. The VPN agent for Windows and Mac for this project were available upon configuring the portal to an accessible IP address within the test network environment. Lastly, there is an official GlobalProtect app available on the Google Play Store by Palo Alto Networks for the Android to utilize. However, for the Android to connect to the VPN through the app, the active Palo Alto firewall must be authenticated with a license key. The ova file was provided for this project from the beginning, but no license key as its absence won't entirely affect the network performance of the VPN.

The steps of configuring a Palo Alto firewall VM are straightforward upon learning where to look for the necessary features to configure. The default credentials to access its web GUI uses the word "admin" as both the username and password, but for a FIPS-CC move Palo

Alto firewall, the password is “paloalto” instead. Many steps in setting the firewall are available on the Palo Alto Network’s live community website, but the necessary steps to configure the firewall are listed here.

The firewall should be installed from a provided ova file into VMware ESXi or a similar virtual machine hypervisor. Once the installation is finished, open the console to begin configuring the VM by providing a static IP address to be used for the management interface. It can be assigned 192.168.42.10 as the GlobalProtect portal is going to use 192.168.42.2. In the console, enter configure mode by typing in “configure” and tapping the Enter button. Then enter these following commands, substituting the tags with necessary information:

- set deviceconfig system type static
- set deviceconfig system ip-address <ip-address> netmask <netmask> default-gateway <default-gateway> dns-setting servers primary <dns-servers>
- commit
- exit

Verify the network information with “show system info” before opening an Internet browser and visiting the firewall’s GUI via the static IP address entered from the previous steps. After logging in with the default credentials, the first simple step is to create self-signed chain of certificates to use for security profiles for later configurations, as well as export to the Windows and Mac clients to save as a trusted source. This is done by selecting the Device tab and clicking the Certificate Manager on the side option bar. Select the Generate button below to begin creating the root certificate; fill in the Certificate and Common Name fields, to check Certificate Authority, and proceed to generate. The intermediate certificate follows the same steps except the Signed By dropdown box should select the root certificate. Lastly, the server certificate to be

used in the SSL/TLS Profile should have its Common Name specify the designated portal IP address, which would be 192.168.42.2, not have the Certificate Authority box checked, and have an IP = 192.168.42.2 Certificate Attribute field added for client devices to trust. Commit changes frequently after making a significant addition or edit to the firewall configuration.

The next major step in configuring the Palo Alto firewall would be to set up its virtual ethernet interfaces so that one interface can serve as the GlobalProtect VPN portal while another ethernet interface can operate as the 192.168.43.0 gateway for the desktop server. Beforehand, two interface management profiles, one Layer 3 trust zone, and one virtual router should be created for the virtual ethernets to use. These can all be found upon selecting the Network tab on top while the options are found along the side menu. The trust zone can be created in the Zone option with the only configuration outside of naming it is to select Layer 3 as its Type before creating it. Starting with the Interface Management option, create a profile with permitted services ping, ssh, https, and response pages and name it “allow-mgt”. Create a second profile with only ping as the permitted service with the name “allow-ping” to signify its limited rights. In the Virtual Routers option, create a virtual router with the only necessary configuration being in the Static Routes tab. Add a static route for the virtual router to use by default, defining it with IPv4 address 0.0.0.0/0 as the destination and using GlobalProtect portal address 192.168.42.2 as the next hop.

After creating the three services and committing the changes, select the Interfaces option while still in the Network section of the Palo Alto firewall. Select a virtual ethernet, Ethernet 1/1 for example, to configure into the 192.168.43.0 gateway. This and the second ethernet, Ethether 1/2, are both Layer 3 types. Set the virtual router and zone to the router and zone created from the previous steps. Input 192.168.43.1/24 into the IPv4 tab. On the Advanced tab, select “allow-

ping" for the interface management before accepting the configurations. Perform the same steps with Ethernet 1/2, but use 192.168.42.2/24 for the IPv4 section as this interface is to be used for the GlobalProtect portal.

The final major step is to configure the GlobalProtect gateway and portal, both located on the side menu while in the Network tab. Create the portal by implementing the main interface to be Ethernet 1/2 and its static IP range, 192.168.42.2/24 on the General tab. In the Authentication tab, select the SSL/TLS Profile created earlier and create an entry for Client Authentication using local authentication. Under the Agent tab, create the agent config; within the config window, check the box to generate and accept cookies while setting the root certificate to encrypt/decrypt the cookies. Go to Add External to input the portal IP address, 192.168.42.2. Lastly, go to App and select "On-Demand" for manual portal login. The gateway configuration, located next to the portal option on the side menu, follows the same configuration as the portal. The Agent tab for the gateway, however, should have tunnel mode and IPsec enabled, add in the IP range 192.168.0.0 – 192.168.255.255 for Client Settings, and input a range in the IP pool for the firewall to pull from for clients. The range can be provided in the 192.168.43.0 network, an example being 192.168.43.20 – 192.168.43.50.

Commit the changes for the configurations to take effect. To add a user to login to the portal upon visiting 192.168.42.2 on a web browser, go to Device and select Local User Database to manually add a username and password to login to the web portal. The portal page, upon successfully logging in, allows GlobalProtect VPN agents for Windows and Mac to be downloaded and used to connect to the VPN freely.

For a Palo Alto VM to switch from its normal organization mode to FIPS and Common Criteria (FIPS-CC) support, the configuration must be done on the VM's terminal on a VMware

application, such as the ESXi's main webpage or a console via the VMware vSphere Client software. A precaution to note regarding switching to FIPS-CC mode causes the VM to reset to factory settings, which is why a second Palo Alto firewall VM is recommended for this purpose while alternating with the first firewall in normal mode during testing. The mode switch is done by inputting "debug system maintenance-mode" into the console to boot up the Maintenance Recovery Tool (MRT). The MRT interface then shows a selection of choices, one of which is to set FIPS-CC mode. The process is overall straightforward and takes a few minutes for the change to complete, depending on how much space within the virtual environment is available. Once FIPS-CC mode is fully operational, it can be configured the same way as a normal mode firewall can using virtually the same steps as mentioned.

Regarding the OpenVPN server, it was within scope to create it from scratch using an Ubuntu Server 17.04 VM. The OpenVPN server is relatively easy to setup, given that there are plenty step-by-step setup guides available online to follow, such as the guide provided by Digital Ocean. Although the webpage recommended the setup for Ubuntu 16.04, it was still applicable on the 17.04 version. OpenVPN openly supports SSL as its main implementation instead of IPsec or PPTP due to portability, ease of configuration, and compatibility with NAT and dynamic addresses. OpenVPN utilizes User Datagram Protocol (UDP) packets, which uses checksums for data integrity but is otherwise connectionless as to reach the destination endpoint without being slowed by handshakes and other overhead accumulating processes compared to TCP. However, specific configuration can permit OpenVPN to use TCP port 443, the commonly used port for secure tunneling. This OpenVPN also uses the AES-256 cipher instead of the AES-128 the website recommends mainly because the base.conf file has the 256 cipher in place from

installation. Check the Appendix section at the end for files to edit and configure for OpenVPN use.

For OpenVPN to utilize FIPS, OpenSSL and the FIPS Object Module are required as they work conjointly. It would be best to build the OpenSSL module alongside the FIPS Object Module as the former needs to be compatible for the latter to force the server to use FIPS-approved encryption algorithms. The FIPS Object Module can be downloaded via the wget command from the OpenSSL website and unpackage with the gunzip and tar commands:

- wget https://www.openssl.org/source/openssl-fips-2.0.16.tar.gz
- gunzip -c openssl-fips-2.0.16.tar.gz | tar xf –

Once the tar.gz file is unpackaged, enter the newly created openssl-fips-2.0.16 folder to begin configuring the FIPS module (OpenSSL Software Foundation, 2017). Run the config script in the folder to build up the module for the VM to use.

Repeat the process in downloading an OpenSSL file from the same website, but with the package openssl-1.0.2o.tar.gz instead when reusing the commands. The OpenSSL community has stated that the 2.0 FIPS module is fully compatible with either the 1.0.1 or 1.0.2 versions of OpenSSL, meaning it is safe to continue installing the module by running the config file following its unpackaging. Upon full installation of OpenSSL, check the version the VM environment is using with the "openssl version" command. If the system returns "fips" in its version, example being OpenSSL 1.0.2o-fips 23 followed by the date of installation, then the system can enable FIPS mode. Set environment variable "OPENSSL_FIPS=1" by the export command; the 1 variable enables FIPS while 0 turns it off. The test to check that FIPS is working under OpenSSL is to check the hash of a file with MD5 using the command "openssl md5 [file.txt]". MD5 is not FIPS-approved, meaning the system should return an error setting the

digest. SHA1, on the other hand, is FIPS-approved, meaning that the FIPS module is enforcing the system as it should when enabled.

The applications of setting up the OpenVPN client endpoints differed with each platform. Windows 10 required installing the client from the OpenVPN website, creating the config folder in the C:\Program Files\OpenVPN file path, and utilizing a File Transfer Protocol (FTP) application to drag the opvn file from the Ubuntu Server into the config folder. Mac OS required the installation of the open-sourced Tunnelblick application to perform OpenVPN tasks and like Windows, requires the opvn file to be dropped into Tunnelblick to establish the VPN connection. However, in the context of this research, for Tunnelblick to accept the configuration file, the ta.key file was required to be in the same directory as the dragged opvn file.

The process on Ubuntu Linux is straightforward via Terminal by typing in the "sudo apt-get update" and "sudo apt-get install openvpn" commands. The opvn file, in this case, can be dropped anywhere as long as the "sudo openvpn --config [filename].opvn" can be executed to establish the connection. The Android platform has both the OpenVPN Connect and OpenVPN for Android apps to be a suitable client to use the opvn file and successfully connect to the server. It should be noted that the former requires the same instruction as Mac's Tunnelblick in including both the opvn file and the ta.key file in the same directory to allow the Android to connect to the VPN server, at least within the parameters of this research, while the latter is able to work with simply importing the opvn file alone.

Lastly, the VM running PPTP is installed through Windows's Server Manager. The VPN is installed through the Remote Access server role with the DirectAccess and VPN (RAS) service. Upon installing the features, click on the following link for the Getting Started Wizard to configure the server, starting with the Deploy VPN Only option. The Routing and Remote

Access Management Console opens, wherein the newly deployed server's configuration can be customizable. The only service to enable in the customization is VPN access. Upon finishing the steps and starting the Routing and Remote Access service, right-click on the server to access Properties, head to the IPv4 tab, and add a range of IP addresses into the static address pool for the VPN to assign clients with. Considering that the VPN VM is within the 192.168.42.0 subnet, it is acceptable to input a static range within the 192.168.43.0 subnet.

Enabling and disabling FIPS on Windows Server is also a straightforward process, done by opening the Local Security Policy console and navigating on the side directory through Local Policies and Security Options. In the Policy table, locate "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" and right-click on it to access Properties. Select Enabled or Disabled, apply the change, and restart the VM for it to take effect.

With PPTP as a legacy protocol, most devices can utilize it by registering the VPN connection through a device's respective wireless and network settings. One glaring exception to this would be Mac's capacity to cooperate with Windows applications. Certain Mac operating systems, such as the macOS Sierra through later patches, have removed the option to form a PPTP VPN connection due to PPTP's lack of defined security settings. As a remedy to this issue, there are VPN software available online, ranging from free open-sourced to purchasable enterprise applications with free trial periods, that can permit the mentioned Mac OSs to form a PPTP connection. Available applications include Flow VPN, Shimo, and VPN Tracker; Flow VPN is free and open-sourced while the last two provide free thirty-day trial periods before requiring a paid subscription to use all features.

3.3 RH-2: Bandwidth Differences

With FAVE's development and the configurations to test for each VPN server, there is the matter of understanding the underlying factors that affect the bandwidth and transfer rate results. On top of testing network speed and connectivity between device endpoints for each VPN server, security must be enforced on personal devices. The iperf results provide benchmarks to determine the tuning parameters necessary in upholding the ideal balance between speed and security.

3.3.1 Device Information

The following information relays the hardware and configurations used for this project in forming the framework. The first half details the router, hard drives, VMs, and their software/firmware configurations. The router serves as the default gateway for the 192.168.42.0 subnet, one hard drive hosts the VMware ESXi holding the VPN VMs, and a second hard drive is set aside as the desktop hosting the iperf server agent.

- ASUS RT-AC68U Wireless Router, 5GHz, IP address 192.168.42.1
- HP ProLiant DL165 G6 Server Basic SATA with 4 Proliant DL165 G6 Storage Drives, RAID 10, SecurityOnion Ubuntu 14.04.5
- Dell OptiPlex 9020 with 4 CPUs x Intel® Core™ i5-4670 CPU @ 3.40 GHz, 15.91 GB memory with ESXi v6.5.0-standard, IP address 192.168.42.4
    - Palo Alto v8.0.5
    - Ubuntu Server 17.04 hosting OpenVPN
    - Windows Server 2016 hosting PPTP VPN service

- Dell OptiPlex 9020 x64-based PC hard drive hosting Windows Server 2016, Intel64 Family 6 Model 60 Stepping 3 GenuineIntel ~800 Mhz processor, IP address 192.168.43.250

For the second hard drive that was meant to serve as the iperf server agent to operate in the 192.168.43.0 network, the above VPN VMs were configured to permit open virtual ethernet interfaces to operate as the 43 subnet's default gateway. The VPNs, active one at a time with the other VMs suspended, were assigned 192.168.42.2 as the local static IP while their additional virtual ethernets were assigned 192.168.43.1. The two hard drives were hardwired into a switch to allow a firm connection for this network relation. As for the client endpoint devices involved in this research, they include the following hardware and operating system information.

- Microsoft Surface Book with Windows 10 Professional x64 based OS, Intel64 Family 6 Model 78 Stepping 3 GenuineIntel ~2396 MHz processor, 8118 MB physical memory, 9398 MB virtual memory, and five NICs.
- MacOS Sierra Version 10.12.3, iMac (27-inch, Mid 2010), 2.93 GHz Intel Core i7 processor, 16 GB evenly split into four 1333 MHz DDR3 memory modules, and an ATI Radeon HD 5750 1024 MB graphics card.
- Samsung Galaxy S7 Edge (Verizon) [SM-G935V], Android v7.0 OS, kernel version 3.18.31, hardware version REV0.7, 2.1 GHz Qualcomm MSM8996 Snapdragon 820 processor, 4.0 GB memory RAM, 32 GB storage, and four security software applications.

3.3.2 Tuning Factors

The purpose of FAVE is to define VPN parameters according to the needs of a network entity, hence determine favored settings. The needs generally scale between wanting a speedy

response between network communications to encrypting every data packet passing through for extra protection. VPNs would have to accommodate to a good variety of client devices just if the clients themselves can establish connections to the VPN they are signing into. The framework is essentially mindful of all participants, given the settings are adjusted accordingly.

There is a checklist of tuning factors to watch out for. These items include network topology, device arrangement, choice of encryption, and partitioned system resources among clients. For repeatability, it is prudent for the researcher or network administrator to maintain a level of consistency upon setting up a framework like FAVE to test different VPN structures.

The layout of the framework's networks must be drawn out to clearly understand the packet machinations between server and client. This includes participating subnets, IP address assignments, and defined gateways. The hardware aspect of forming a VPN-tuning framework is also significant in the sense that the server devices and main router would need to be hardwired together while preserving the subnet schema. This also applies to personal devices representing the client side of the topology as their participation suggests the necessary configurations for VPNs to allow clients of different platforms and operating systems to join in the first place.

Within a VPN itself lies the possibilities of implementing settings such as firewall rules and encryption algorithms. While the use of firewalls would permit or deny certain IP addresses and ports, the focus lies upon choice of encryption. The choices of encryption can fall upon the provided defaults recommended by the VPN make or follow the encryption algorithms strictly enforced under FIPS settings.

Lastly, the VPNs themselves would need to be given an equal share of resources in whichever virtual environment they are stationed in for the framework to provide tuning insights, though it is recommended to use VMware ESXi to store and manage the VPN VMs due to its

ease of use. This is because integral details such as CPU memory, core processors, and related can affect VPN VM performance, such as increased bandwidth within short intervals of time because of the VPN struggling to forward the packets between client and server in a sensible manner.

4 RESULTS & ANALYSIS

FAVE consists of two primary components, the iperf tool collecting network traffic data from different platforms upon connecting to different VPN architectures and the security measures derived from data analysis by my own research and expertise compared to a few best practices currently publicized regarding the hardening of personal devices. This chapter explains in detail the construction of this framework's mechanics and the results brought about running the iperf tool to collect the VPN traffic performance for each machine involved.

4.1 Data Collection

The collection of data has been manually harvested through executing the iperf server and client agents between the four main platforms: Windows 10, macOS Sierra, and Android mobile. Data from the default settings of each VPN server, as well as under FIPS-defined settings when applicable, has been collected. At different intervals during the research period, the devices had established connections with the following VPN setups:

- Palo Alto GlobalProtect using IPsec and AES-256 encryption.
- OpenVPN using SSL and AES-256 encryption on an Ubuntu Server 17.04 VM.
- A Windows Server 2016 VM running a PPTP server.

The raw data collection is in the following subsection while the analysis of the collected data plus calculated standard deviation, specifically Figure 4-1 and Figure 4-2, is in the next section of the chapter.

4.1.1 Fixed Transfer Bandwidth

To determine any significant changes in network speeds between these VPNs, a benchmark is provided in iperf sending a fixed transfer of 100 MBytes over connections with no connections to a VPN and with connections through the featured VPNs. This is performed by tcpdump through the Windows client device to monitor the bytes during communication, ensuring that they are going over the VPN. Typically, the client address would be viewed using TCP port 5001, iperf's default communication, but when logged into a VPN, the communicating port is expected to be TCP port 443, or UDP port 1194 for OpenVPN using UDP traffic. See Table 4.1 to view produced bytes per frame produced by each fixed file.

Table 4.1: Fixed Transfer Results

Setting	Interval	Transfer	Bandwidth	Bytes per Frame
No VPN	0.0 – 4.0 sec	100 MBytes	211 Mbits/sec	106 MBytes
Palo Alto, No FIPS	0.0 – 6.2 sec	100 MBytes	136 Mbits/sec	118 MBytes
Palo Alto, FIPS	0.0 – 846.7 sec	100 MBytes	993 Kbits/sec	117 MBytes
OpenVPN, No FIPS	0.0 – 13.2 sec	100 MBytes	85.6 Mbits/sec	125 MBytes
OpenVPN, FIPS	0.0 – 13.3 sec	100 MBytes	82.1 Mbits/sec	122 MBytes
PPTP, No FIPS	0.0 – 11.1 sec	100 MBytes	75.7 Mbits/sec	115 MBytes
PPTP, FIPS	0.0 – 14.0 sec	100 MBytes	60.0 Mbits/sec	115 MBytes

4.1.2 Windows Client Results

Performance tests on a Windows 10 client have revealed strong consistency between all twenty ten-second intervals for each setting. See Table 4.2 for the first iperf test communicating to the server within the isolated network's reach, no VPN involved or active during the two-hundred seconds the testing took place. See Table 4.3 for the iperf tests on the Palo Alto VMs, differentiated by the fact that one VM is in normal mode while the other is in FIPS-CC mode. See Table 4.4 for the iperf tests conducted on OpenVPN with OpenSSL FIPS Object Module disabled and enabled. See Table 4.5 for the iperf tests results done through the Windows PPTP server with the FIPS policy disabled and enabled.

Table 4.2: Windows Client Iperf Results with No VPN

Interval	Transfer	Bandwidth
0.0-10.0 sec	241 MBytes	202 Mbits/sec
10.0-20.0 sec	238 MBytes	200 Mbits/sec
20.0-30.0 sec	240 MBytes	202 Mbits/sec
30.0-40.0 sec	240 MBytes	201 Mbits/sec
40.0-50.0 sec	243 MBytes	204 Mbits/sec
50.0-60.0 sec	241 MBytes	202 Mbits/sec
60.0-70.0 sec	240 MBytes	201 Mbits/sec
70.0-80.0 sec	238 MBytes	200 Mbits/sec
80.0-90.0 sec	235 MBytes	197 Mbits/sec
90.0-100.0 sec	232 MBytes	195 Mbits/sec
100.0-110.0 sec	234 MBytes	197 Mbits/sec
110.0-120.0 sec	234 MBytes	197 Mbits/sec
120.0-130.0 sec	260 MBytes	218 Mbits/sec
130.0-140.0 sec	262 MBytes	219 Mbits/sec
140.0-150.0 sec	260 MBytes	218 Mbits/sec
150.0-160.0 sec	272 MBytes	228 Mbits/sec
160.0-170.0 sec	266 MBytes	223 Mbits/sec
170.0-180.0 sec	263 MBytes	221 Mbits/sec
180.0-190.0 sec	264 MBytes	221 Mbits/sec
190.0-200.0 sec	266 MBytes	223 Mbits/sec
	Total	**Average**
0.0-200.0 sec	4.85 GBytes	208 Mbits/sec

Table 4.3: Windows Client Iperf Results Using Palo Alto VPN

	No FIPS		FIPS	
Interval	**Transfer**	**Bandwidth**	**Transfer**	**Bandwidth**
0.0-10.0 sec	160 MBytes	135 Mbits/sec	1.00 MBytes	839 Kbits/sec
10.0-20.0 sec	169 MBytes	142 Mbits/sec	1.00 MBytes	839 Kbits/sec
20.0-30.0 sec	171 MBytes	143 Mbits/sec	768 KBytes	629 Kbits/sec
30.0-40.0 sec	174 MBytes	146 Mbits/sec	640 KBytes	524 Kbits/sec
40.0-50.0 sec	177 MBytes	149 Mbits/sec	512 KBytes	419 Kbits/sec
50.0-60.0 sec	170 MBytes	143 Mbits/sec	1.25 MBytes	1.05 Mbits/sec
60.0-70.0 sec	164 MBytes	137 Mbits/sec	1.12 MBytes	944 Kbits/sec
70.0-80.0 sec	171 MBytes	144 Mbits/sec	1.25 MBytes	1.05 Mbits/sec
80.0-90.0 sec	172 MBytes	144 Mbits/sec	1.12 MBytes	944 Kbits/sec
90.0-100.0 sec	159 MBytes	133 Mbits/sec	1.12 MBytes	944 Kbits/sec
100.0-110.0 sec	168 MBytes	141 Mbits/sec	1.38 MBytes	1.15 Mbits/sec
110.0-120.0 sec	169 MBytes	142 Mbits/sec	1.12 MBytes	944 Kbits/sec
120.0-130.0 sec	171 MBytes	144 Mbits/sec	1.25 MBytes	1.05 Mbits/sec
130.0-140.0 sec	176 MBytes	147 Mbits/sec	1.38 MBytes	1.15 Mbits/sec
140.0-150.0 sec	167 MBytes	140 Mbits/sec	1.12 MBytes	944 Kbits/sec
150.0-160.0 sec	172 MBytes	144 Mbits/sec	1.12 MBytes	944 Kbits/sec
160.0-170.0 sec	176 MBytes	148 Mbits/sec	1.12 MBytes	944 Kbits/sec
170.0-180.0 sec	163 MBytes	137 Mbits/sec	1.12 MBytes	944 Kbits/sec
180.0-190.0 sec	170 MBytes	143 Mbits/sec	1.12 MBytes	944 Kbits/sec
190.0-200.0 sec	169 MBytes	142 Mbits/sec	1.25 MBytes	1.05 Mbits/sec
	Total	**Average**	**Total**	**Average**
0.0-200.0 sec	3.31 GBytes	142 Mbits/sec	21.8 MBytes	906 Kbits/sec

Table 4.4: Windows Client Iperf Results Using OpenVPN

	No FIPS		FIPS	
Interval	**Transfer**	**Bandwidth**	**Transfer**	**Bandwidth**
0.0-10.0 sec	117 MBytes	98.5 Mbits/sec	107 MBytes	88.5 Mbits/sec
10.0-20.0 sec	122 MBytes	103 Mbits/sec	110 MBytes	90.5 Mbits/sec
20.0-30.0 sec	118 MBytes	99.3 Mbits/sec	110 MBytes	90.0 Mbits/sec
30.0-40.0 sec	122 MBytes	102 Mbits/sec	102 MBytes	92.6 Mbits/sec
40.0-50.0 sec	123 MBytes	103 Mbits/sec	101 MBytes	91.4 Mbits/sec
50.0-60.0 sec	111 MBytes	93.0 Mbits/sec	111 MBytes	92.7 Mbits/sec
60.0-70.0 sec	106 MBytes	88.6 Mbits/sec	122 MBytes	92.1 Mbits/sec
70.0-80.0 sec	108 MBytes	90.5 Mbits/sec	110 MBytes	90.5 Mbits/sec
80.0-90.0 sec	112 MBytes	93.7 Mbits/sec	110 MBytes	92.7 Mbits/sec
90.0-100.0 sec	119 MBytes	100 Mbits/sec	109 MBytes	90.0 Mbits/sec
100.0-110.0 sec	110 MBytes	92.7 Mbits/sec	107 MBytes	89.8 Mbits/sec
110.0-120.0 sec	115 MBytes	96.7 Mbits/sec	110 MBytes	91.7 Mbits/sec
120.0-130.0 sec	107 MBytes	89.8 Mbits/sec	112 MBytes	93.7 Mbits/sec
130.0-140.0 sec	122 MBytes	102 Mbits/sec	106 MBytes	88.6 Mbits/sec
140.0-150.0 sec	114 MBytes	95.8 Mbits/sec	108 MBytes	89.8 Mbits/sec
150.0-160.0 sec	118 MBytes	99.3 Mbits/sec	110 MBytes	90.9 Mbits/sec
160.0-170.0 sec	122 MBytes	102 Mbits/sec	110 MBytes	91.5 Mbits/sec
170.0-180.0 sec	123 MBytes	103 Mbits/sec	107 MBytes	87.3 Mbits/sec
180.0-190.0 sec	120 MBytes	101 Mbits/sec	110 MBytes	91.3 Mbits/sec
190.0-200.0 sec	121 MBytes	102 Mbits/sec	109 MBytes	89.4 Mbits/sec
	Total	**Average**	**Total**	**Average**
0.0-200.0 sec	2.28 GBytes	97.8 Mbits/sec	2.58 GBytes	90.7 Mbits/sec

Table 4.5: Windows Client Iperf Results Using Windows Server VPN (PPTP)

	No FIPS		FIPS	
Interval	**Transfer**	**Bandwidth**	**Transfer**	**Bandwidth**
0.0-10.0 sec	105 MBytes	88.1 Mbits/sec	119 MBytes	99.5 Mbits/sec
10.0-20.0 sec	92.6 MBytes	77.7 Mbits/sec	112 MBytes	94.0 Mbits/sec
20.0-30.0 sec	98.2 MBytes	82.4 Mbits/sec	116 MBytes	97.0 Mbits/sec
30.0-40.0 sec	108 MBytes	90.4 Mbits/sec	120 MBytes	101 Mbits/sec
40.0-50.0 sec	103 MBytes	86.4 Mbits/sec	112 MBytes	94.3 Mbits/sec
50.0-60.0 sec	101 MBytes	84.7 Mbits/sec	111 MBytes	93.4 Mbits/sec
60.0-70.0 sec	93.2 MBytes	78.2 Mbits/sec	109 MBytes	91.5 Mbits/sec
70.0-80.0 sec	105 MBytes	88.1 Mbits/sec	109 MBytes	91.4 Mbits/sec
80.0-90.0 sec	110 MBytes	92.5 Mbits/sec	113 MBytes	94.9 Mbits/sec
90.0-100.0 sec	112 MBytes	94.2 Mbits/sec	93.4 MBytes	78.3 Mbits/sec
100.0-110.0 sec	115 MBytes	96.5 Mbits/sec	94.9 MBytes	79.6 Mbits/sec
110.0-120.0 sec	111 MBytes	93.1 Mbits/sec	108 MBytes	90.3 Mbits/sec
120.0-130.0 sec	116 MBytes	97.5 Mbits/sec	102 MBytes	85.8 Mbits/sec
130.0-140.0 sec	99.8 MBytes	83.7 Mbits/sec	104 MBytes	87.0 Mbits/sec
140.0-150.0 sec	80.6 MBytes	67.6 Mbits/sec	103 MBytes	86.2 Mbits/sec
150.0-160.0 sec	97.1 MBytes	81.5 Mbits/sec	124 MBytes	104 Mbits/sec
160.0-170.0 sec	106 MBytes	89.1 Mbits/sec	110 MBytes	91.9 Mbits/sec
170.0-180.0 sec	98.4 MBytes	82.5 Mbits/sec	123 MBytes	103 Mbits/sec
180.0-190.0 sec	114 MBytes	95.5 Mbits/sec	116 MBytes	97.3 Mbits/sec
190.0-200.0 sec	106 MBytes	89.0 Mbits/sec	117 MBytes	98.5 Mbits/sec
	Total	**Average**	**Total**	**Average**
0.0-200.0 sec	2.02 GBytes	86.9 Mbits/sec	2.16 GBytes	92.9 Mbits/sec

4.1.3 Mac Client Results

The network performance tests on the macOS Sierra client yielded significantly strong consistency in its intervals for nearly all settings. See Table 4.6 for the iperf test communicating to the server agent with no VPN involved. See Table 4.7 for the iperf tests on the Palo Alto VMs. See Table 4.8 for the iperf tests conducted on OpenVPN. See Table 4.9 for the iperf tests results done through the Windows PPTP server.

Table 4.6: Mac Client Iperf Results with No VPN

Interval	Transfer	Bandwidth
0.0-10.0 sec	220 MBytes	184 Mbits/sec
10.0-20.0 sec	220 MBytes	184 Mbits/sec
20.0-30.0 sec	220 MBytes	185 Mbits/sec
30.0-40.0 sec	219 MBytes	184 Mbits/sec
40.0-50.0 sec	219 MBytes	184 Mbits/sec
50.0-60.0 sec	220 MBytes	185 Mbits/sec
60.0-70.0 sec	221 MBytes	185 Mbits/sec
70.0-80.0 sec	219 MBytes	184 Mbits/sec
80.0-90.0 sec	216 MBytes	181 Mbits/sec
90.0-100.0 sec	218 MBytes	183 Mbits/sec
100.0-110.0 sec	217 MBytes	182 Mbits/sec
110.0-120.0 sec	220 MBytes	184 Mbits/sec
120.0-130.0 sec	202 MBytes	169 Mbits/sec
130.0-140.0 sec	202 MBytes	169 Mbits/sec
140.0-150.0 sec	202 MBytes	170 Mbits/sec
150.0-160.0 sec	201 MBytes	168 Mbits/sec
160.0-170.0 sec	199 MBytes	167 Mbits/sec
170.0-180.0 sec	202 MBytes	169 Mbits/sec
180.0-190.0 sec	199 MBytes	167 Mbits/sec
190.0-200.0 sec	203 MBytes	170 Mbits/sec
	Total	**Average**
0.0-200.0 sec	4.14 GBytes	178 Mbits/sec

Table 4.7: Mac Client Iperf Results Using Palo Alto VPN

	No FIPS		FIPS	
Interval	**Transfer**	**Bandwidth**	**Transfer**	**Bandwidth**
0.0-10.0 sec	192 MBytes	161 Mbits/sec	768 KBytes	629 Kbits/sec
10.0-20.0 sec	191 MBytes	160 Mbits/sec	768 KBytes	629 Kbits/sec
20.0-30.0 sec	192 MBytes	161 Mbits/sec	640 KBytes	524 Kbits/sec
30.0-40.0 sec	194 MBytes	163 Mbits/sec	640 KBytes	524 Kbits/sec
40.0-50.0 sec	188 MBytes	158 Mbits/sec	768 KBytes	629 Kbits/sec
50.0-60.0 sec	193 MBytes	162 Mbits/sec	896 KBytes	734 Kbits/sec
60.0-70.0 sec	196 MBytes	164 Mbits/sec	1.00 MBytes	839 Kbits/sec
70.0-80.0 sec	196 MBytes	164 Mbits/sec	1.00 MBytes	839 Kbits/sec
80.0-90.0 sec	194 MBytes	163 Mbits/sec	1.12 MBytes	944 Kbits/sec
90.0-100.0 sec	190 MBytes	159 Mbits/sec	1.00 MBytes	839 Kbits/sec
100.0-110.0 sec	195 MBytes	163 Mbits/sec	1.00 MBytes	839 Kbits/sec
110.0-120.0 sec	194 MBytes	163 Mbits/sec	1.12 MBytes	944 Kbits/sec
120.0-130.0 sec	193 MBytes	162 Mbits/sec	1.12 MBytes	944 Kbits/sec
130.0-140.0 sec	193 MBytes	162 Mbits/sec	1.12 MBytes	944 Kbits/sec
140.0-150.0 sec	196 MBytes	164 Mbits/sec	1.00 MBytes	839 Kbits/sec
150.0-160.0 sec	198 MBytes	166 Mbits/sec	1.12 MBytes	944 Kbits/sec
160.0-170.0 sec	196 MBytes	164 Mbits/sec	1.00 MBytes	839 Kbits/sec
170.0-180.0 sec	196 MBytes	165 Mbits/sec	1.00 MBytes	839 Kbits/sec
180.0-190.0 sec	194 MBytes	163 Mbits/sec	1.12 MBytes	944 Kbits/sec
190.0-200.0 sec	194 MBytes	163 Mbits/sec	1.12 MBytes	944 Kbits/sec
	Total	**Average**	**Total**	**Average**
0.0-200.0 sec	3.78 GBytes	162 Mbits/sec	19.4 MBytes	810 Kbits/sec

Table 4.8: Mac Client Iperf Results Using OpenVPN

	No FIPS		FIPS	
Interval	**Transfer**	**Bandwidth**	**Transfer**	**Bandwidth**
0.0-10.0 sec	98.9 MBytes	82.9 Mbits/sec	92.9 MBytes	77.9 Mbits/sec
10.0-20.0 sec	104 MBytes	87.1 Mbits/sec	79.4 MBytes	66.6 Mbits/sec
20.0-30.0 sec	92.4 MBytes	77.5 Mbits/sec	80.0 MBytes	67.1 Mbits/sec
30.0-40.0 sec	92.8 MBytes	77.8 Mbits/sec	84.5 MBytes	70.9 Mbits/sec
40.0-50.0 sec	96.2 MBytes	80.7 Mbits/sec	78.9 MBytes	66.2 Mbits/sec
50.0-60.0 sec	92.8 MBytes	77.8 Mbits/sec	80.1 MBytes	67.2 Mbits/sec
60.0-70.0 sec	92.6 MBytes	77.7 Mbits/sec	87.9 MBytes	73.7 Mbits/sec
70.0-80.0 sec	98.4 MBytes	82.5 Mbits/sec	79.1 MBytes	66.4 Mbits/sec
80.0-90.0 sec	92.5 MBytes	77.6 Mbits/sec	79.8 MBytes	66.9 Mbits/sec
90.0-100.0 sec	92.6 MBytes	77.7 Mbits/sec	113 MBytes	95.0 Mbits/sec
100.0-110.0 sec	98.2 MBytes	82.4 Mbits/sec	79.9 MBytes	67.0 Mbits/sec
110.0-120.0 sec	92.2 MBytes	77.4 Mbits/sec	79.6 MBytes	66.8 Mbits/sec
120.0-130.0 sec	93.0 MBytes	78.0 Mbits/sec	83.2 MBytes	69.8 Mbits/sec
130.0-140.0 sec	96.8 MBytes	81.2 Mbits/sec	80.4 MBytes	67.4 Mbits/sec
140.0-150.0 sec	93.4 MBytes	78.3 Mbits/sec	79.8 MBytes	66.9 Mbits/sec
150.0-160.0 sec	92.4 MBytes	77.5 Mbits/sec	81.8 MBytes	68.6 Mbits/sec
160.0-170.0 sec	99.2 MBytes	83.3 Mbits/sec	85.6 MBytes	71.8 Mbits/sec
170.0-180.0 sec	92.6 MBytes	77.7 Mbits/sec	82.0 MBytes	68.8 Mbits/sec
180.0-190.0 sec	92.6 MBytes	77.7 Mbits/sec	88.6 MBytes	74.3 Mbits/sec
190.0-200.0 sec	99.0 MBytes	83.0 Mbits/sec	79.2 MBytes	66.5 Mbits/sec
	Total	**Average**	**Total**	**Average**
0.0-200.0 sec	1.86 GBytes	79.8 Mbits/sec	1.64 GBytes	70.3 Mbits/sec

Table 4.9: Mac Client Iperf Results Under Windows Server VPN (PPTP)

	No FIPS		FIPS	
Interval	**Transfer**	**Bandwidth**	**Transfer**	**Bandwidth**
0.0-10.0 sec	152 MBytes	128 Mbits/sec	72.5 MBytes	60.8 Mbits/sec
10.0-20.0 sec	134 MBytes	112 Mbits/sec	72.0 MBytes	60.4 Mbits/sec
20.0-30.0 sec	74.9 MBytes	62.8 Mbits/sec	182 MBytes	152 Mbits/sec
30.0-40.0 sec	69.8 MBytes	58.5 Mbits/sec	218 MBytes	183 Mbits/sec
40.0-50.0 sec	79.0 MBytes	66.3 Mbits/sec	222 MBytes	186 Mbits/sec
50.0-60.0 sec	73.9 MBytes	62.0 Mbits/sec	218 MBytes	183 Mbits/sec
60.0-70.0 sec	125 MBytes	105 Mbits/sec	84.6 MBytes	71.0 Mbits/sec
70.0-80.0 sec	220 MBytes	185 Mbits/sec	72.9 MBytes	61.1 Mbits/sec
80.0-90.0 sec	218 MBytes	182 Mbits/sec	76.8 MBytes	64.4 Mbits/sec
90.0-100.0 sec	218 MBytes	183 Mbits/sec	85.1 MBytes	71.4 Mbits/sec
100.0-110.0 sec	222 MBytes	186 Mbits/sec	192 MBytes	161 Mbits/sec
110.0-120.0 sec	219 MBytes	183 Mbits/sec	223 MBytes	187 Mbits/sec
120.0-130.0 sec	197 MBytes	165 Mbits/sec	193 MBytes	162 Mbits/sec
130.0-140.0 sec	105 MBytes	88.2 Mbits/sec	199 MBytes	167 Mbits/sec
140.0-150.0 sec	72.5 MBytes	60.8 Mbits/sec	204 MBytes	171 Mbits/sec
150.0-160.0 sec	74.8 MBytes	62.7 Mbits/sec	194 MBytes	163 Mbits/sec
160.0-170.0 sec	91.8 MBytes	77.0 Mbits/sec	80.6 MBytes	67.6 Mbits/sec
170.0-180.0 sec	74.9 MBytes	62.8 Mbits/sec	81.8 MBytes	68.6 Mbits/sec
180.0-190.0 sec	74.0 MBytes	62.1 Mbits/sec	85.5 MBytes	71.7 Mbits/sec
190.0-200.0 sec	152 MBytes	128 Mbits/sec	77.1 MBytes	64.7 Mbits/sec
	Total	**Average**	**Total**	**Average**
0.0-200.0 sec	2.59 GBytes	111 Mbits/sec	2.77 GBytes	119 Mbits/sec

4.1.4 Android Client Results

This section lists the iperf results gathered from the Android platform. See Table 4.10 for the iperf test communicating to the server agent with no VPN involved. See Table 4.11 for the iperf tests on the Palo Alto VMs. See Table 4.12 for the iperf tests conducted on OpenVPN. See Table 4.13 for the iperf tests results done through the Windows PPTP server.

An important reminder regarding the analysis of Palo Alto through Android is that for the Android to connect to its VPN, the firewall requires licensing for the app to successfully connect. The normal mode firewall was the only Palo Alto VM to gain a license key to permit the

connection, meaning results for the FIPS-CC mode firewall couldn't be attained during the testing period. However, a request for a license key was sent out to a Palo Alto Networks associate, but time constraints forced FIPS-CC mode results to be cut for this scenario.

Table 4.10: Android Client Iperf Results with No VPN

Interval	**Transfer**	**Bandwidth**
0.0-10.0 sec	73.6 MBytes	61.8 Mbits/sec
10.0-20.0 sec	78.5 MBytes	65.9 Mbits/sec
20.0-30.0 sec	80.4 MBytes	67.4 Mbits/sec
30.0-40.0 sec	79.0 MBytes	66.3 Mbits/sec
40.0-50.0 sec	76.0 MBytes	63.8 Mbits/sec
50.0-60.0 sec	71.9 MBytes	60.3 Mbits/sec
60.0-70.0 sec	71.1 MBytes	59.7 Mbits/sec
70.0-80.0 sec	79.2 MBytes	66.5 Mbits/sec
80.0-90.0 sec	78.1 MBytes	65.5 Mbits/sec
90.0-100.0 sec	76.4 MBytes	64.1 Mbits/sec
100.0-110.0 sec	77.5 MBytes	65.0 Mbits/sec
110.0-120.0 sec	77.5 MBytes	65.0 Mbits/sec
120.0-130.0 sec	67.6 MBytes	56.7 Mbits/sec
130.0-140.0 sec	74.8 MBytes	62.7 Mbits/sec
140.0-150.0 sec	80.9 MBytes	67.8 Mbits/sec
150.0-160.0 sec	77.0 MBytes	64.6 Mbits/sec
160.0-170.0 sec	78.0 MBytes	65.4 Mbits/sec
170.0-180.0 sec	76.5 MBytes	64.2 Mbits/sec
180.0-190.0 sec	69.8 MBytes	58.5 Mbits/sec
190.0-200.0 sec	71.5 MBytes	60.0 Mbits/sec
	Total	**Average**
0.0-200.0 sec	1.48 GBytes	63.6 Mbits/sec

Table 4.11: Android Client Iperf Results Using Palo Alto VPN

	No FIPS	
Interval	**Transfer**	**Bandwidth**
0.0-10.0 sec	71.5 MBytes	60.0 Mbits/sec
10.0-20.0 sec	63.8 MBytes	53.5 Mbits/sec
20.0-30.0 sec	59.4 MBytes	49.8 Mbits/sec
30.0-40.0 sec	65.5 MBytes	54.9 Mbits/sec
40.0-50.0 sec	61.5 MBytes	51.6 Mbits/sec
50.0-60.0 sec	67.1 MBytes	56.3 Mbits/sec
60.0-70.0 sec	66.0 MBytes	55.4 Mbits/sec
70.0-80.0 sec	60.1 MBytes	50.4 Mbits/sec
80.0-90.0 sec	59.4 MBytes	49.8 Mbits/sec
90.0-100.0 sec	62.5 MBytes	52.4 Mbits/sec
100.0-110.0 sec	63.9 MBytes	53.6 Mbits/sec
110.0-120.0 sec	66.4 MBytes	55.7 Mbits/sec
120.0-130.0 sec	58.6 MBytes	49.2 Mbits/sec
130.0-140.0 sec	63.8 MBytes	53.5 Mbits/sec
140.0-150.0 sec	66.1 MBytes	55.5 Mbits/sec
150.0-160.0 sec	63.5 MBytes	53.3 Mbits/sec
160.0-170.0 sec	66.6 MBytes	55.9 Mbits/sec
170.0-180.0 sec	63.8 MBytes	53.5 Mbits/sec
180.0-190.0 sec	63.0 MBytes	52.8 Mbits/sec
190.0-200.0 sec	63.6 MBytes	53.4 Mbits/sec
	Total	**Average**
0.0-200.0 sec	1.25 GBytes	53.5 Mbits/sec

Table 4.12: Android Client Iperf Results Using OpenVPN

	No FIPS		FIPS	
Interval	**Transfer**	**Bandwidth**	**Transfer**	**Bandwidth**
0.0-10.0 sec	20.5 MBytes	18.3 Mbits/sec	11.5 MBytes	9.65 Mbits/sec
10.0-20.0 sec	17.0 MBytes	14.9 Mbits/sec	8.75 MBytes	7.34 Mbits/sec
20.0-30.0 sec	17.4 MBytes	15.2 Mbits/sec	7.75 MBytes	6.50 Mbits/sec
30.0-40.0 sec	21.2 MBytes	18.8 Mbits/sec	7.25 MBytes	6.08 Mbits/sec
40.0-50.0 sec	19.2 MBytes	17.3 Mbits/sec	14.8 MBytes	12.4 Mbits/sec
50.0-60.0 sec	25.5 MBytes	21.8 Mbits/sec	17.6 MBytes	14.8 Mbits/sec
60.0-70.0 sec	22.7 MBytes	19.2 Mbits/sec	20.5 MBytes	17.2 Mbits/sec
70.0-80.0 sec	20.2 MBytes	18.4 Mbits/sec	23.9 MBytes	20.0 Mbits/sec
80.0-90.0 sec	23.6 MBytes	19.9 Mbits/sec	22.8 MBytes	19.1 Mbits/sec
90.0-100.0 sec	20.1 MBytes	18.0 Mbits/sec	24.1 MBytes	20.2 Mbits/sec
100.0-110.0 sec	24.0 MBytes	20.1 Mbits/sec	18.1 MBytes	15.2 Mbits/sec
110.0-120.0 sec	25.5 MBytes	21.8 Mbits/sec	24.4 MBytes	20.4 Mbits/sec
120.0-130.0 sec	23.5 MBytes	19.8 Mbits/sec	24.5 MBytes	20.6 Mbits/sec
130.0-140.0 sec	22.6 MBytes	19.1 Mbits/sec	23.6 MBytes	19.8 Mbits/sec
140.0-150.0 sec	21.5 MBytes	18.9 Mbits/sec	23.5 MBytes	19.7 Mbits/sec
150.0-160.0 sec	21.2 MBytes	18.7 Mbits/sec	23.2 MBytes	19.5 Mbits/sec
160.0-170.0 sec	21.3 MBytes	18.8 Mbits/sec	24.2 MBytes	20.3 Mbits/sec
170.0-180.0 sec	24.5 MBytes	20.4 Mbits/sec	19.5 MBytes	16.4 Mbits/sec
180.0-190.0 sec	30.8 MBytes	25.5 Mbits/sec	19.4 MBytes	16.3 Mbits/sec
190.0-200.0 sec	28.1 MBytes	23.6 Mbits/sec	25.0 MBytes	21.0 Mbits/sec
	Total	**Average**	**Total**	**Average**
0.0-200.0 sec	450 MBytes	19.4 Mbits/sec	384 MBytes	16.1 Mbits/sec

Table 4.13: Android Client Iperf Results Using Windows Server VPN (PPTP)

	No FIPS		FIPS	
Interval	**Transfer**	**Bandwidth**	**Transfer**	**Bandwidth**
0.0-10.0 sec	33.9 MBytes	28.4 Mbits/sec	41.5 MBytes	34.8 Mbits/sec
10.0-20.0 sec	31.8 MBytes	26.6 Mbits/sec	26.1 MBytes	21.9 Mbits/sec
20.0-30.0 sec	32.1 MBytes	26.9 Mbits/sec	29.9 MBytes	25.1 Mbits/sec
30.0-40.0 sec	28.4 MBytes	23.8 Mbits/sec	25.2 MBytes	21.2 Mbits/sec
40.0-50.0 sec	15.8 MBytes	13.2 Mbits/sec	21.8 MBytes	18.2 Mbits/sec
50.0-60.0 sec	27.5 MBytes	23.1 Mbits/sec	35.9 MBytes	30.1 Mbits/sec
60.0-70.0 sec	34.0 MBytes	28.5 Mbits/sec	31.0 MBytes	26.0 Mbits/sec
70.0-80.0 sec	22.6 MBytes	19.0 Mbits/sec	41.4 MBytes	34.7 Mbits/sec
80.0-90.0 sec	29.5 MBytes	24.7 Mbits/sec	23.9 MBytes	20.0 Mbits/sec
90.0-100.0 sec	21.6 MBytes	18.1 Mbits/sec	25.9 MBytes	21.7 Mbits/sec
100.0-110.0 sec	19.5 MBytes	16.4 Mbits/sec	18.8 MBytes	15.7 Mbits/sec
110.0-120.0 sec	17.8 MBytes	14.9 Mbits/sec	41.5 MBytes	34.8 Mbits/sec
120.0-130.0 sec	28.1 MBytes	23.6 Mbits/sec	27.9 MBytes	23.4 Mbits/sec
130.0-140.0 sec	18.2 MBytes	15.3 Mbits/sec	37.4 MBytes	31.4 Mbits/sec
140.0-150.0 sec	32.5 MBytes	27.3 Mbits/sec	24.8 MBytes	20.8 Mbits/sec
150.0-160.0 sec	21.4 MBytes	17.9 Mbits/sec	25.6 MBytes	21.5 Mbits/sec
160.0-170.0 sec	33.6 MBytes	28.2 Mbits/sec	27.1 MBytes	22.8 Mbits/sec
170.0-180.0 sec	30.2 MBytes	25.4 Mbits/sec	22.6 MBytes	19.0 Mbits/sec
180.0-190.0 sec	34.2 MBytes	28.7 Mbits/sec	49.8 MBytes	41.7 Mbits/sec
190.0-200.0 sec	25.1 MBytes	21.1 Mbits/sec	47.5 MBytes	39.8 Mbits/sec
	Total	**Average**	**Total**	**Average**
0.0-200.0 sec	538 MBytes	22.6 Mbits/sec	626 MBytes	26.1 Mbits/sec

4.2 Framework Analysis

The validity of FAVE relied on the network connectivity and traffic between iperf client and server through VPN tunneling. This section covers the analysis and understanding of how the data resulted in the way it did, particularly the technological issues and limitations during the research period resulting in questionable outputs at first glance. The objectives include evaluating the framework's capability of providing distinction of different encryption settings implemented upon VPNs and how it affections different client platforms reaching out to the same desktop endpoint.

4.2.1 Limitations

The main limitation during the research involved acquiring sufficient resources to build the framework with. Given that FAVE was built from scratch, it took a good amount of time to plan out the network topology and the machines needed to support the topology. The computer lab the framework development took place in spared the needed machines for the framework, which included a wireless router and a couple of Dell hard drives to install VMware ESXi and Windows Server 2016. The ova and iso files for the VPN VMs were also provided and promptly installed into the ESXi environment with limited space. Given equal partitioning of memory and space for each VM, the ESXi environment was able to hold up to five VMs.

The limited space on the ESXi also signified that only a few snapshots in total could be taken when a need arose to roll back a VM. If there were more snapshots saved that the ESXi could allocate, any suspended VM would be unable to power on, outputting "Module 'MonitorLoop" power on failed" on the interface. Only a couple of snapshots should be taken as failsafe using sound judgment if experimenting with a VPN configuration with a high error risk. Only when the VPN configuration becomes satisfactory should the lingering snapshots be deleted to free up additional space for the ESXi environment.

Concerning further hardware limitations, packet monitoring and capturing was initially performed through the ASUS Wireless Router using tcpdump. The methodology switched over to monitoring through the ProLiant DL165 G6 Basic SATA because it was discovered that the router was dropping about a quarter of the expected data packet total during the fixed transfer iperf tests. The Basic SATA boasted a more robust frame in keeping the data flowing with a zero percent drop rate.

Another major limitation was the liberal usage of Palo Alto software to certain degrees. The ova file to create a Palo Alto firewall was provided by the computer lab due to the sponsorship with Palo Alto Networks. However, obtaining a license key to use the firewall's full features, including the ability for the GlobalProtect app for Android to connect to it, required a more formal process. One license key was readily available from the research lab to use for the first Palo Alto firewall set in normal mode. A request was made for another license key to use for the second firewall set in FIPS-CC mode to fully test the Android's interoperability over their settings and further validate the framework by comparing averaged bandwidth rates, but due to time constraints, this factor had to be cut from the analysis.

4.2.2 VPN Traffic Explanation

As the primary objective of the framework, VPN tuning is the process of boosting the performance of a VPN after an evaluation of its architecture and hardware. Following the series of iperf tests on each client device running through different VPN settings, a distinct trend can be found shared across all the platforms during the twenty consecutive intervals.

The tests identified the majority average of bandwidth of VPNs set to FIPS mode to be lower than the bandwidth of VPNs using their default security configurations, including the bandwidth for client devices regularly connecting to the iperf server under no VPN tunnel. However, there are certain exceptions wherein the FIPS-enabled settings of a VPN resulted in a higher averaged bandwidth than the vendor-default settings. How significantly the bandwidth rate differs between the two varies depending on what VPN is involved and which client device attempted the connection. See Figures 4-1 and 4-2 for graphs depicting the differences via

standard error bars; the gap in difference between a VPN not using FIPS and using FIPS can be disregarded if the error bar endpoints are within each other's range.

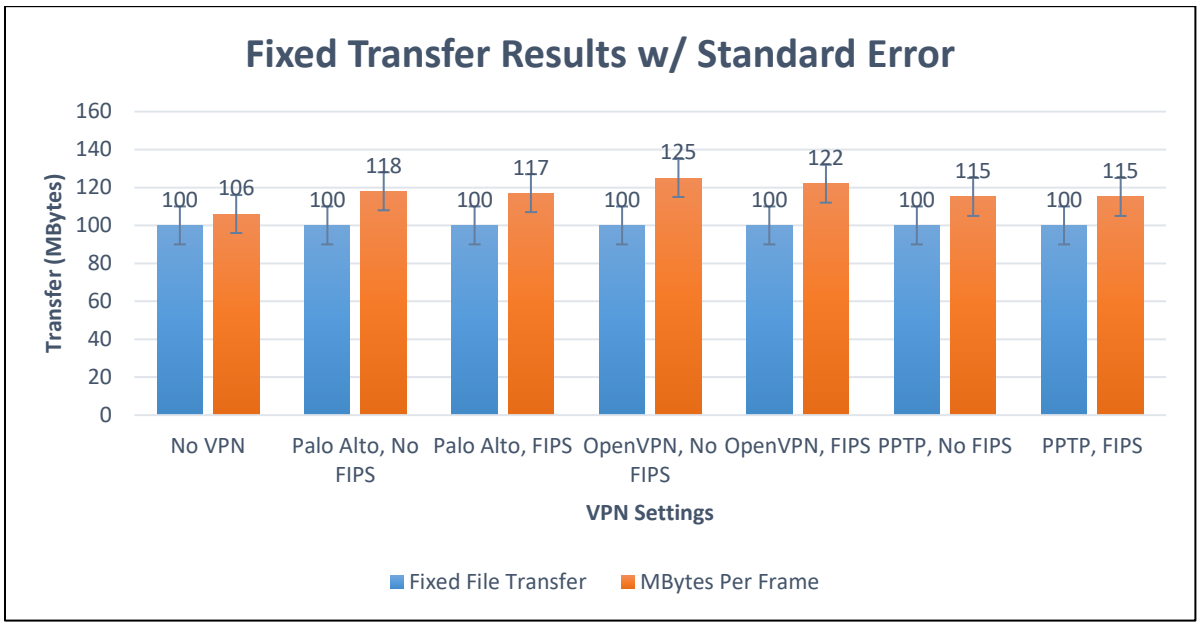

Figure 4-1: Fixed Transfer Results Graph with Standard Error Bars

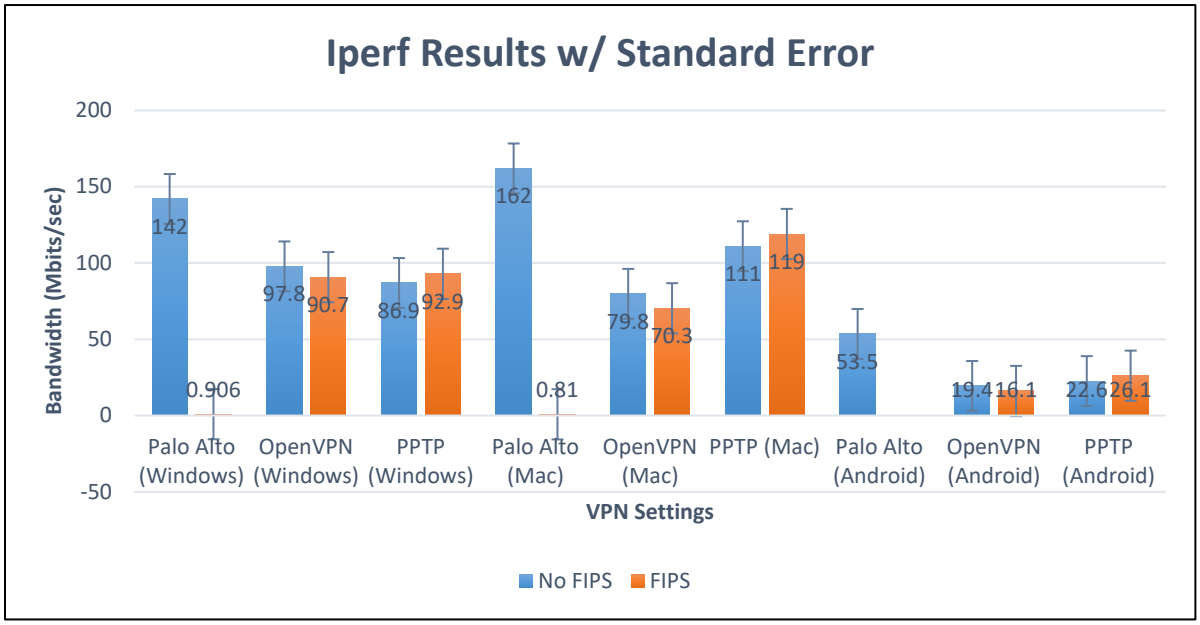

Figure 4-2: Collected Iperf Results Graph with Standard Error Bars

A glaring example of bandwidth rates distinctively separate on the same VPN setting is the Palo Alto setting as seen on both Windows and Mac clients. An example outcome from analyzing the results of Figure 4-2 can be an administrator leaning away from using Palo Alto if FIPS 140-2 was the designated security standards for the network, preferring to use OpenVPN instead because of its considerably balanced rate between network speed and secure connection, at least at first glance. There is still the fact that OpenVPN has incurred the highest encryption overhead among the others in the fixed byte transfer tests. A more thorough explanation is required on why the averaged bandwidths are the way they are, which would involve deeper investigation into the security and encryption processes occurring during client-server communication.

To ensure confidence regarding the bandwidth differences between FIPS and no-FIPS, t-tests have been taken for each client and VPN. The t-tests are held to the recognized significance level of 0.05. They were calculated by Microsoft Excel, using the arrays of collected FIPS and non-FIPS iperf results, one-tailed distribution, and depending on the variance for each array, equal or unequal types. Table 4.14 lists the p-values that meet significance level standards.

Table 4.14: T-Test Significant P-values

VPN & Client	**T-Test P-value**
Palo Alto, Windows	3.63037E-31
OpenVPN, Windows	1.64965E-07
PPTP, Windows	0.006129484
Palo Alto, Mac	9.93778E-39
OpenVPN, Mac	1.74438E-06
OpenVPN, Android	0.006316539
PPTP, Android	0.040322866

The missing factors, PPTP, Mac and Palo Alto, Android, are excluded due to the latter's limitations and the former having a p-value higher than the significance level. Mac's bandwidth behavior through the PPTP connection shows no difference regardless of VPN settings and is explained further down this section. As for the listed p-values, the statistics reaffirm the degrees of variance between FIPS and no-FIPS for the selected VPN and client factors. The most significant p-values involve Palo Alto on both featured clients followed by OpenVPN for Windows and Mac, further contributing to the bandwidth gaps between these two VPNs.

During the initial setup of the FIPS Object Module for OpenVPN, it was found that for FIPS to cooperate with OpenSSL, the product had to be of certain versions. The OpenSSL version used is 1.0.2o. In short, the previous OpenSSL version installed in the Ubuntu Server had to be uninstalled while a compatible version had to be found and installed in its place. This, among a few basic terminal lines of unpackaging and running make files to integrate the FIPS Object Module and compatible OpenSSL version together took nearly two months to complete. This is because they are separate applications that aren't provided in a Linux machine by default, although in some cases a version of OpenSSL comes preinstalled. Compared to the way a Palo Alto firewall switches into FIPS-CC mode within several minutes of instruction and waiting, as written in the Methodology chapter, this is a troubling issue as there is the expectation that the FIPS-CC firewall is performing multiple encryption policies and procedures, but even then, there isn't a one-hundred percent guarantee that the transferred data is fully protected from outside exploitation.

In addition to the presented findings, these results would otherwise confirm the idea of consistent encryption overhead effecting the bandwidth rate by significantly slowing its performance, meaning that FIPS mode enforces only strong and approved encryption algorithms

to protect the network as processed by previous network trials regarding the effects of encryption (Anitha Rani, Ram Kumar, & Prem Kumar, 2016). The average bandwidth of each client sans VPN usage is significantly high, attributing to the lack of encryption overhead, though there is some deviance between results to take note of between the sequential iperf tests. Standard deviations were taken for each client and setting to differentiate the results. See Table 4.15 for the calculated standard deviation for this setting.

Table 4.15: Standard Deviations for Clients' Iperf Results Using No VPN

Operating System	**Standard Deviation σ**
Windows	10.924628140124
Mac	7.4973328590906
Android	3.0181782584864

The Palo Alto bandwidth while in FIPS-CC mode is significantly lower in comparison to its normal counterpart upon inspection of Tables 4.3 and 4.7 throughout all clients that connected to Palo Alto with and without FIPS upon comparing their resulting averages. However, in the case of the Android client, there is no available comparison between FIPS and no FIPS settings as it was previously mentioned that the license key needed to activate the full features of the Palo Alto firewall running FIPS-CC mode was requested for, but it was not issued within the time frame. Given that this VPN participant is a sophisticated specialized software built by an enterprise for this purpose, the iperf tests validate its efficiency with general consistency between intervals, at least when under normal mode. See Table 4.16 for the standard deviations between Palo Alto's iperf bandwidth results.

Table 4.16: Standard Deviations of Palo Alto Iperf Results

Setting	Standard Deviation σ
Windows, No FIPS	4.0570925550202
Windows, FIPS	184.79964826806
Mac, No FIPS	1.9104973174543
Mac, FIPS	141.26305249427
Android, No FIPS	2.5602490113268
Android, FIPS	N/A

It would be safe to determine from the standard deviations of the collective iperf results under FIPS-CC mode for each available client device, as well as the fixed transfer byte test performed in this setting, that heavy encryption procedures and TCP handshakes are taking place within the VPN tunnel during live communication.

The iperf results for OpenVPN appear to also follow the trend of the bandwidth average being noticeably greater using its default settings versus the average collected with the FIPS module enabled. However, unlike Palo Alto, there isn't a significant gap between the two settings, only separable by less than ten megabytes compared to markedly crossing over from megabytes to kilobytes. This would suggest that there isn't a huge difference between averaged bandwidth rates, but the t-tests listed in Table 4.14 for clients connecting under OpenVPN do suggest significant differences between the two settings on each client. Compared to the Palo Alto and PPTP settings, the OpenVPN t-test p-values are in the middle range with PPTP p-values reaching the closest to default significance level 0.05. Nonetheless, the individual variances for OpenVPN settings running for each client are low. See Table 4.17 for the standard deviations of OpenVPN's iperf results.

Table 4.17: Standard Deviations of OpenVPN Iperf Results

Setting	Standard Deviation σ
Windows, No FIPS	4.7156627317907
Windows, FIPS	1.5847712768725
Mac, No FIPS	2.7795503233437
Mac, FIPS	6.4733221764408
Android, No FIPS	2.4116125310671
Android, FIPS	4.9398352958373

OpenVPN's low bandwidth is attributed to the overhead of encrypting small data packets instead of one large packet at a time (Hoekstra & Musulin, 2011). On a related note, while performing the fixed transfer test on Windows under both OpenVPN settings, the readout during packet capture revealed that not all one-hundred megabytes were filtered despite no dropped packets. The phenomenon was discovered to be the Windows client innately compressing the data packet through some internal system process. Data compression reduces frame sizes being transmitted over network links. The exact process or feature in Windows that caused the compression, inadvertently skewing the packet conversation analysis, is unknown, but discovering this phenomenon alone using the FAVE framework is within scope, fulfilling its purpose.

It appears that it is this same property that has affected the iperf continuous interval and fixed transfer results for the Windows Server VPN. Aaron Margosis of Microsoft discussed about the company's revoking of their previous recommendation of enabling FIPS mode for all versions of Windows client and Windows Server. The article reminds that enabling FIPS mode would result in disabling non-validated cryptographic classes, suggesting that in the case of Windows it is doing more harm than good. Since the trend of lower bandwidth from the testing

leans toward the fact that secure encryption overhead is incurring, there indicates the possibility that the FIPS policy may be disabling some of the system's more commonly used algorithms because they haven't been validated.

The resulting averages from the Windows Server VPN overall appear to be the inverse of the recurring trend with the FIPS-enabled averages having higher bandwidth than the averages obtained with the FIPS option disabled. The iperf results appear to be particularly spread out, especially for the Mac client. See Table 4.18 for the standard deviations of the Windows Server VPN iperf results.

Table 4.18: Standard Deviations of Windows Server PPTP Iperf Results

Setting	Standard Deviation σ
Windows, No FIPS	7.2209608086459
Windows, FIPS	6.7732174776837
Mac, No FIPS	50.379955339401
Mac, FIPS	53.405470459495
Android, No FIPS	5.022688523092
Android, FIPS	7.3748966094448

Discussions within the Apple community shared similar issues regarding the Mac platform's peculiar behavior and cooperability with a Windows VPN, particularly the later versions gaining slow throughput upon connecting with a PPTP VPN. The macOS Sierra Version 10.12.3 and later versions beyond 10.12 has Apple drop the option to establish a PPTP link because of the protocol's insecurity, requiring third party VPN software to create connectivity with a PPTP VPN. However, even then, the stability of third party software

attempting to reintroduce a dropped feature is expected cause some internal conflict with the system processes because of conflicting objectives.

Nonetheless, the utilization of FAVE revealed the issue and with additional analysis, appropriate action would be taken in considering the VPN style and recommended client devices to use for optimal performances and secure communication.

4.2.3 FIPS Application

The decision to use FIPS 140-2 as a standard to compare against default security settings of different VPN structures proved to be satisfactory. FIPS 140-2 was initially selected due to it being the currently published standard of cryptographic protection to test against, helped by the fact that the encryption algorithms it forces systems to use are approved by the federal government for computer security.

Its actual application on real world devices, however, is dependent on the user's needs and desires. While the application of FIPS has proven to slow down bandwidth for the most part or is considered redundant in the Windows case, it is still considered secure with its set of encryption algorithms.

4.2.4 Framework Validation

The FAVE framework has accomplished its purpose in providing a unique method of tuning VPN security settings through the evaluation of iperf performance test results. By reading the trends between continuous intervals and calculating the average and standard deviation of the results, explanations regarding anomalies within the bandwidth behavior can determine the true cause.

Overall, it is integral for an organization or entity to first understand the infrastructure of the kind of VPN it wants to deploy prior to a formal setup. Mapping out the VPN topology, such as determining how much bandwidth the VPN can afford for connections before all users would experience bottleneck, would save a lot of time and resources early during the trial run. For example, solutions for preserving bandwidth if the organization does not wish to pay for additional bandwidth can include cutting down the number of clients allowed to connect to the VPN. In any case, it takes manual effort to monitor VPN performance and make the appropriate patches to maintain speed and security.

5 CONCLUSION & FUTURE WORK

The purpose of this research was to develop a VPN tuning framework from scratch with the default settings of each involved VPN infrastructure compared to an enforced standard of encryption algorithms published by the US federal government. The main objective to validate the framework for future continuation involves being able to determine sources of bandwidth issue from comparing a VPN's normal settings with FIPS-enforced rules. The research question tied in to the objective was to determine the differences between a VPN using FIPS-enforced settings compared to its default security. This chapter summarizes the findings and opens future possibilities using this research as the basis.

FAVE is the developed VPN tuning framework, depicting the roles of setup and roles of tested VPNs during the client-server communication. It would be safe to say that the framework is repeatable, whether under the same circumstances as performed in this research or for a whole new VPN-testing scenario altogether. It detected significant differences between different VPN settings as discussed previously in the Framework Analysis section of the last chapter. Of course, given the limited resources, frequent technological issues, and time constraints faced in this undertaking, the framework is far from perfect, but it does allow plenty of room for improvement.

5.1 Future Research

The following section provides additional insights and discarded ideas that can be considered for future research:

- Inclusion of Linux and additional platforms as clients, including the cut Android running through Palo Alto FIPS-CC mode.
- Automated performance testing through scripts.

5.1.1 Linux Involvement

After careful contemplation over the framework development, Linux was not used as a client platform despite using OpenVPN on an Ubuntu Server. Include Linux in all performance tests by installing the operating system into a desktop machine instead of having it as a VM.

Linux was initially going to be used alongside Windows, Mac, and Android, but for the framework to apply, Linux needed to connect to all VPNs with no issue. One problem found was configuring an Ubuntu Linux VM to conform to the Palo Alto Network's GlobalProtect through the strongSwan client, which led to narrowing the scope of this research by cutting most of Linux's planned involvement out of the framework due to its unforeseen difficulty and lack of time.

The technical issues behind GlobalProtect had extended to the mobile app to a degree during testing. Because the framework was developed by machines and software provided by the BYU CSRL, there were limited features to test with. Fortunately, much progress was capable of being carried out without a full Palo Alto subscription, but this is a note for a future that to make full use of an enterprise VPN software, it will require a planned budget and a sure dedication to study its performance.

The developed framework has opened the possibility to involve more platforms and operating systems for future VPN testing, such as iOS in addition to Linux and possibly its different distros. Ubuntu would be ideal to test given its simplicity, but there's also the possibility of testing the likes of Debian, Fedora, or even CentOS if one researcher feels daring enough.

5.1.2 Automated Tests

The VPN performance tests were executed manually through the iperf v2 tool, which was required to be installed in all actively participating devices. Could have there been a more efficient method in collecting network performance data through a single executable command? This question poses the possibility for researchers to write up a script under any coding language, such as C, Ruby, Node JS, or Python, and have the script run through the motions of collecting the data without the hassle of going back and forth between devices.

In addition to writing scripts that can perform automated network performance analyses, the challenge lies in making the script compatible with a variety of tools like iperf. The automated scripts themselves would have to be open-sourced as to be interoperable with any tool and platform available. This consideration was formulated in mind after considering certain tools are not even backwards compatible, iperf being the forefront example. Iperf was selected on the basis that it was the only viable network testing tool that can cooperate with Windows, Mac, Android, and Linux all at once. Other open-sourced network tools like netpipe and netperf are compatible with Linux and most UNIX-based distributions, but I haven't found any documentation of these tools able to operate in other platforms like Windows without requiring intensive time-consuming configuration.

5.2 Observations

The framework that was developed to tune VPNs according to network performance on devices is enough to prove the need to accommodate to popular opinion. It should be brought to attention that the major issue discovered while developing FAVE was the manner of configuring FIPS to each participating VPN. For the Palo Alto firewall and Windows Server, setting up FIPS took a duration of ten minutes or less due to their systems having a capacity to enable FIPS within a few steps. OpenVPN, however, required more attention in configuring its settings to cooperate with the FIPS Object Module.

As written in the Methodology chapter and explained in the Framework Analysis section, the ability to configure FIPS instead of simply switching it off or on grants complete control over the system. The duration of finding and manually installing the FIPS Object Module and compatible OpenSSL version extended to nearly two months. Given that OpenVPN and the involved extensions are open-sourced, there is the assumption that freedom of control was what allowed the average bandwidth of OpenVPN with FIPS to reach up to the non-FIPS results better than Palo Alto, providing a suggestion for deeper investigation behind the latter's security.

To further validate the efficiency of the VPN-tuning framework with the potential to expand and improve upon it, it would be helpful to have users sufficiently knowledgeable in the information technology field, such as the students and faculty involved with the BYU Information Technology program and the Cyber Security Research Lab, to test and scrutinize the tuning framework for themselves by either with the tools and systems used here or by similar methods. The additional validation will provide necessary insight that haven't been covered from the initial testing performed from this research, opening new paths and possibility on VPN performance tuning for the future to come.

REFERENCES

Alshalan, A., Pisharody, S., & Huang, D. (2016). A Survey of Mobile VPN Technologies. *IEEE Communications Surveys and Tutorials*, *18*(2), 1177–1196. https://doi.org/10.1109/COMST.2015.2496624

Anitha Rani, N. R., Ram Kumar, S. K., & Prem Kumar, P. (2016). A Survey on Data Redundancy Check in a Hybrid Cloud by using Convergent Encryption. *Indian Journal of Science and Technology*, *9*(4), 1–5. https://doi.org/10.17485/ijst/2016/v9i4/87036

Aouini, I., Ben Azzouz, L., & Saidane, L. A. (2016). A secure neighborhood area network using IPsec. *2016 International Wireless Communications and Mobile Computing Conference, IWCMC 2016*, 102–107. https://doi.org/10.1109/IWCMC.2016.7577041

Ashraf, Z., & Yousaf, M. (2016). SECURE INTER-VLAN IPv6 ROUTING : IMPLEMENTATION & EVALUATION, *28*(3), 3007–3014.

Barker, W. C., & Barker, E. (2012). Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher: NIST Special Publication 800-67, Revision 2.

Caddy, T. (2005). FIPS 140-2. In H. C. A. van Tilborg (Ed.), *Encyclopedia of Cryptography and Security* (pp. 227–230). Boston, MA: Springer US. https://doi.org/10.1007/0-387-23483-7_168

Gallenmüller, S., Emmerich, P., Wohlfart, F., Raumer, D., & Carle, G. (2015). Comparison of frameworks for high-performance packet IO. *ANCS 2015 - 11th 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 29–38. https://doi.org/10.1109/ANCS.2015.7110118

Heinemann, C., Chaduvu, S. shankar, Byerly, A., & Uskov, A. (2016). OpenCL and CUDA Software Implementations of Encryption / Decryption Algorithms for IPsec VPNs, 765–770.

Hirschler, B., & Sauter, T. (2016). Performance impact of IPsec in resource-limited smart grid communication. *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, *2016–June*. https://doi.org/10.1109/WFCS.2016.7496517

Hoekstra, B., & Musulin, D. (2011). Comparing TCP performance of tunneled and non-tunneled traffic using OpenVPN.

"How to Set Up an OpenVPN Server on Ubuntu 16.04", *Tutorials Digital Ocean Community Web*, [online] Available: https://www.digitalocean.com/community/tutorials/how-to-set-up-an-openvpn-server-on-ubuntu-16-04.

Jin, Y., Tomoishi, M., & Matsuura, S. (2016). Enhancement of VPN authentication using GPS information with geo-privacy protection. *2016 25th International Conference on Computer Communications and Networks, ICCCN 2016*. https://doi.org/10.1109/ICCCN.2016.7568518

Kolahi, S. S., Cao, Y., & Chen, H. (2013). Impact of SSL Security o n Bandwidth and D elay in IEEE 802 . 11n WLAN Using Windows 7, 1–4. https://doi.org/10.1109/CSNDSP.2016.7574043

Liyanage, M., Ylianttila, M., & Gurtov, A. (2016). Improving the tunnel management performance of secure VPLS architectures with SDN. *2016 13th IEEE Annual Consumer Communications and Networking Conference, CCNC 2016*, 530–536. https://doi.org/10.1109/CCNC.2016.7444836

Margosis, A (2014). "Why We're Not Recommending 'FIPS mode' Anymore." Accessed 18 March 2018. https://blogs.technet.microsoft.com/secguide/2014/04/07/why-were-not-recommending-fips-mode-anymore/

Narayan, S., Ishrar, S., Kumar, A., Gupta, R., & Khan, Z. (2016). Performance analysis of 4to6 and 6to4 transition mechanisms over point to point and IPSec VPN protocols. https://doi.org/10.1109/WOCN.2016.7759027

OpenSSL Software Foundation. (2017). OpenSSL Cryptography and SSL/TLS Toolkit. *User Guide*, *0*, 1–225. https://doi.org/10.4337/9781782545583.00006

Rao, M., Newe, T., Grout, I., & Mathur, A. (2016). An FPGA-based reconfigurable IPSec AH core with efficient implementation of SHA-3 for high speed IoT applications. *International Journal of Applied Engineering Research*, *9*(22), 5968–5974. https://doi.org/10.1002/sec

Raumer, D., Gallenmuller, S., Emmerich, P., Mardian, L., & Carle, G. (2016). Efficient Serving of VPN Endpoints on COTS Server Hardware. *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, 164–169. https://doi.org/10.1109/CloudNet.2016.25

Roca, V., & Fall, S. (2014). Too big or too small? the PTB-PTS ICMP-based attack against IPsec gateways. *2014 IEEE Global Communications Conference, GLOBECOM 2014*, 530–536. https://doi.org/10.1109/GLOCOM.2014.7036862

Singh, K. K. V. V, & Gupta, H. (2016). A New Approach for the Security of VPN. *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, (13), 11099–11104.

Uskov, A., Byerly, A., & Heinemann, C. (2016). Advanced encryption standard analysis with multimedia data on Intel?? AES-NI architecture. *International Journal of Computer Science and Applications*, *13*(2), 89–105.

Yadav, A. (2016). Security Structure of Vpn : a Survey. *International Journal of Recent Innovation in Engineering and Research*, *1*(1), 19–24.

APPENDICES

APPENDIX A. OPENVPN SERVER CONFIGURED FILES

A.1 ~/openvpn-ca/vars

```
# easy-rsa parameter settings

# NOTE: If you installed from an RPM,
# don't edit this file in place in
# /usr/share/openvpn/easy-rsa --
# instead, you should copy the whole
# easy-rsa directory to another location
# (such as /etc/openvpn) so that your
# edits will not be wiped out by a future
# OpenVPN package upgrade.

# This variable should point to
# the top level of the easy-rsa
# tree.
export EASY_RSA="`pwd`"

#
# This variable should point to
# the requested executables
#
export OPENSSL="openssl"
export PKCS11TOOL="pkcs11-tool"
export GREP="grep"


# This variable should point to
# the openssl.cnf file included
# with easy-rsa.
export KEY_CONFIG=`$EASY_RSA/whichopensslcnf $EASY_RSA`

# Edit this variable to point to
# your soon-to-be-created key
# directory.
#
# WARNING: clean-all will do
# a rm -rf on this directory
# so make sure you define
```

```
# it correctly!
export KEY_DIR="$EASY_RSA/keys"

# Issue rm -rf warning
echo NOTE: If you run ./clean-all, I will be doing a rm -rf on
$KEY_DIR

# PKCS11 fixes
export PKCS11_MODULE_PATH="dummy"
export PKCS11_PIN="dummy"

# Increase this to 2048 if you
# are paranoid.  This will slow
# down TLS negotiation performance
# as well as the one-time DH parms
# generation process.
export KEY_SIZE=2048

# In how many days should the root CA key expire?
export CA_EXPIRE=3650

# In how many days should certificates expire?
export KEY_EXPIRE=3650

# These are the default values for fields
# which will be placed in the certificate.
# Don't leave any of these fields blank.
export KEY_COUNTRY="US"
export KEY_PROVINCE="UT"
export KEY_CITY="Provo"
export KEY_ORG="BrighamYoungUniversity"
export KEY_EMAIL="fperez@byu.edu"
export KEY_OU="CSRL"

# X509 Subject Field
export KEY_NAME="server"

# PKCS11 Smart Card
# export PKCS11_MODULE_PATH="/usr/lib/changeme.so"
# export PKCS11_PIN=1234

# If you'd like to sign all keys with the same Common Name, uncomment
the KEY_CN export below
# You will also need to make sure your OpenVPN server config has the
duplicate-cn option set
# export KEY_CN="CommonName"
```

A.2 /etc/openvpn/server.conf

```
################################################
# Sample OpenVPN 2.0 config file for           #
```

```
# multi-client server.                              #
#                                                   #
# This file is for the server side                  #
# of a many-clients <-> one-server                  #
# OpenVPN configuration.                            #
#                                                   #
# OpenVPN also supports                             #
# single-machine <-> single-machine                 #
# configurations (See the Examples page             #
# on the web site for more info).                   #
#                                                   #
# This config should work on Windows                #
# or Linux/BSD systems.  Remember on                #
# Windows to quote pathnames and use                #
# double backslashes, e.g.:                         #
# "C:\\Program Files\\OpenVPN\\config\\foo.key"     #
#                                                   #
# Comments are preceded with '#' or ';'             #
#####################################################

# Which local IP address should OpenVPN
# listen on? (optional)
;local a.b.c.d

# Which TCP/UDP port should OpenVPN listen on?
# If you want to run multiple OpenVPN instances
# on the same machine, use a different port
# number for each one.  You will need to
# open up this port on your firewall.
port 1194

# TCP or UDP server?
;proto tcp
proto udp

# "dev tun" will create a routed IP tunnel,
# "dev tap" will create an ethernet tunnel.
# Use "dev tap0" if you are ethernet bridging
# and have precreated a tap0 virtual interface
# and bridged it with your ethernet interface.
# If you want to control access policies
# over the VPN, you must create firewall
# rules for the the TUN/TAP interface.
# On non-Windows systems, you can give
# an explicit unit number, such as tun0.
# On Windows, use "dev-node" for this.
# On most systems, the VPN will not function
# unless you partially or fully disable
# the firewall for the TUN/TAP interface.
dev tap0
;dev tun
```

```
# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel if you
# have more than one.  On XP SP2 or higher,
# you may need to selectively disable the
# Windows firewall for the TAP adapter.
# Non-Windows systems usually don't need this.
;dev-node MyTap

# SSL/TLS root certificate (ca), certificate
# (cert), and private key (key).  Each client
# and the server must have their own cert and
# key file.  The server and all clients will
# use the same ca file.
#
# See the "easy-rsa" directory for a series
# of scripts for generating RSA certificates
# and private keys.  Remember to use
# a unique Common Name for the server
# and each of the client certificates.
#
# Any X509 key management system can be used.
# OpenVPN can also use a PKCS #12 formatted key file
# (see "pkcs12" directive in man page).
ca ca.crt
cert server.crt
key server.key  # This file should be kept secret

# Diffie hellman parameters.
# Generate your own with:
#   openssl dhparam -out dh2048.pem 2048
dh dh2048.pem

# Network topology
# Should be subnet (addressing via IP)
# unless Windows clients v2.0.9 and lower have to
# be supported (then net30, i.e. a /30 per client)
# Defaults to net30 (not recommended)
;topology subnet

# Configure server mode and supply a VPN subnet
# for OpenVPN to draw client addresses from.
# The server will take 10.8.0.1 for itself,
# the rest will be made available to clients.
# Each client will be able to reach the server
# on 10.8.0.1. Comment this line out if you are
# ethernet bridging. See the man page for more info.
;server 192.168.43.0 255.255.255.0

# Maintain a record of client <-> virtual IP address
# associations in this file.  If OpenVPN goes down or
# is restarted, reconnecting clients can be assigned
# the same virtual IP address from the pool that was
```

```
# previously assigned.
ifconfig-pool-persist ipp.txt

# Configure server mode for ethernet bridging.
# You must first use your OS's bridging capability
# to bridge the TAP interface with the ethernet
# NIC interface.  Then you must manually set the
# IP/netmask on the bridge interface, here we
# assume 10.8.0.4/255.255.255.0.  Finally we
# must set aside an IP range in this subnet
# (start=10.8.0.50 end=10.8.0.100) to allocate
# to connecting clients.  Leave this line commented
# out unless you are ethernet bridging.
server-bridge 192.168.43.1 255.255.255.0 192.168.43.50 192.168.43.100

# Configure server mode for ethernet bridging
# using a DHCP-proxy, where clients talk
# to the OpenVPN server-side DHCP server
# to receive their IP address allocation
# and DNS server addresses.  You must first use
# your OS's bridging capability to bridge the TAP
# interface with the ethernet NIC interface.
# Note: this mode only works on clients (such as
# Windows), where the client-side TAP adapter is
# bound to a DHCP client.
server-bridge

# Push routes to the client to allow it
# to reach other private subnets behind
# the server.  Remember that these
# private subnets will also need
# to know to route the OpenVPN client
# address pool (10.8.0.0/255.255.255.0)
# back to the OpenVPN server.
;push "route 192.168.10.0 255.255.255.0"
;push "route 192.168.20.0 255.255.255.0"

# To assign specific IP addresses to specific
# clients or if a connecting client has a private
# subnet behind it that should also have VPN access,
# use the subdirectory "ccd" for client-specific
# configuration files (see man page for more info).

# EXAMPLE: Suppose the client
# having the certificate common name "Thelonious"
# also has a small subnet behind his connecting
# machine, such as 192.168.40.128/255.255.255.248.
# First, uncomment out these lines:
;client-config-dir ccd
;route 192.168.40.128 255.255.255.248
# Then create a file ccd/Thelonious with this line:
#   iroute 192.168.40.128 255.255.255.248
```

```
# This will allow Thelonious' private subnet to
# access the VPN.  This example will only work
# if you are routing, not bridging, i.e. you are
# using "dev tun" and "server" directives.

# EXAMPLE: Suppose you want to give
# Thelonious a fixed VPN IP address of 10.9.0.1.
# First uncomment out these lines:
;client-config-dir ccd
;route 10.9.0.0 255.255.255.252
# Then add this line to ccd/Thelonious:
#   ifconfig-push 10.9.0.1 10.9.0.2

# Suppose that you want to enable different
# firewall access policies for different groups
# of clients.  There are two methods:
# (1) Run multiple OpenVPN daemons, one for each
#     group, and firewall the TUN/TAP interface
#     for each group/daemon appropriately.
# (2) (Advanced) Create a script to dynamically
#     modify the firewall in response to access
#     from different clients.  See man
#     page for more info on learn-address script.
;learn-address ./script

# If enabled, this directive will configure
# all clients to redirect their default
# network gateway through the VPN, causing
# all IP traffic such as web browsing and
# and DNS lookups to go through the VPN
# (The OpenVPN server machine may need to NAT
# or bridge the TUN/TAP interface to the internet
# in order for this to work properly).
push "redirect-gateway def1 bypass-dhcp"

# Certain Windows-specific network settings
# can be pushed to clients, such as DNS
# or WINS server addresses.  CAVEAT:
# http://openvpn.net/faq.html#dhcpcaveats
# The addresses below refer to the public
# DNS servers provided by opendns.com.
push "dhcp-option DNS 208.67.222.222"
push "dhcp-option DNS 208.67.220.220"

# Uncomment this directive to allow different
# clients to be able to "see" each other.
# By default, clients will only see the server.
# To force clients to only see the server, you
# will also need to appropriately firewall the
# server's TUN/TAP interface.
;client-to-client
```

```
# Uncomment this directive if multiple clients
# might connect with the same certificate/key
# files or common names.  This is recommended
# only for testing purposes.  For production use,
# each client should have its own certificate/key
# pair.
#
# IF YOU HAVE NOT GENERATED INDIVIDUAL
# CERTIFICATE/KEY PAIRS FOR EACH CLIENT,
# EACH HAVING ITS OWN UNIQUE "COMMON NAME",
# UNCOMMENT THIS LINE OUT.
;duplicate-cn

# The keepalive directive causes ping-like
# messages to be sent back and forth over
# the link so that each side knows when
# the other side has gone down.
# Ping every 10 seconds, assume that remote
# peer is down if no ping received during
# a 120 second time period.
keepalive 10 120

# For extra security beyond that provided
# by SSL/TLS, create an "HMAC firewall"
# to help block DoS attacks and UDP port flooding.
#
# Generate with:
#   openvpn --genkey --secret ta.key
#
# The server and each client must have
# a copy of this key.
# The second parameter should be '0'
# on the server and '1' on the clients.
tls-auth ta.key 0 # This file is secret
key-direction 0

# Select a cryptographic cipher.
# This config item must be copied to
# the client config file as well.
# Note that 2.4 client/server will automatically
# negotiate AES-256-GCM in TLS mode.
# See also the ncp-cipher option in the manpage
cipher AES-256-CBC
auth SHA256

# Enable compression on the VPN link and push the
# option to the client (2.4+ only, for earlier
# versions see below)
;compress lz4-v2
;push "compress lz4-v2"

# For compression compatible with older clients use comp-lzo
```

```
# If you enable it here, you must also
# enable it in the client config file.
;comp-lzo

# The maximum number of concurrently connected
# clients we want to allow.
;max-clients 100

# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
#
# You can uncomment this out on
# non-Windows systems.
user nobody
group nogroup

# The persist options will try to avoid
# accessing certain resources on restart
# that may no longer be accessible because
# of the privilege downgrade.
persist-key
persist-tun

# Output a short status file showing
# current connections, truncated
# and rewritten every minute.
status openvpn-status.log

# By default, log messages will go to the syslog (or
# on Windows, if running as a service, they will go to
# the "\Program Files\OpenVPN\log" directory).
# Use log or log-append to override this default.
# "log" will truncate the log file on OpenVPN startup,
# while "log-append" will append to it.  Use one
# or the other (but not both).
;log         openvpn.log
;log-append  openvpn.log

# Set the appropriate level of log
# file verbosity.
#
# 0 is silent, except for fatal errors
# 4 is reasonable for general usage
# 5 and 6 can help to debug connection problems
# 9 is extremely verbose
verb 3

# Silence repeating messages.  At most 20
# sequential messages of the same message
# category will be output to the log.
;mute 20
```

```
# Notify the client that when the server restarts so it
# can automatically reconnect.
explicit-exit-notify 1
```

A.3 /etc/sysctl.conf

```
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3

##############################################################3
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-
path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration
#  based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1


###################################################################
# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through
# redirection. Some network environments, however, require that these
# settings are disabled so review and enable them as needed.
#
# Do not accept ICMP redirects (prevent MITM attacks)
#net.ipv4.conf.all.accept_redirects = 0
```

```
#net.ipv6.conf.all.accept_redirects = 0
# _or_
# Accept ICMP redirects only for gateways listed in our default
# gateway list (enabled by default)
# net.ipv4.conf.all.secure_redirects = 1
#
# Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#

###################################################################
# Magic system request Key
# 0=disable, 1=enable all
# Debian kernels have this set to 0 (disable the key)
# See https://www.kernel.org/doc/Documentation/sysrq.txt
# for what other values do
#kernel.sysrq=1

###################################################################
# Protected links
#
# Protects against creating or following links under certain
conditions
# Debian kernels have both set to 1 (restricted)
# See https://www.kernel.org/doc/Documentation/sysctl/fs.txt
#fs.protected_hardlinks=0
#fs.protected_symlinks=0
```

A.4 /usr/share/doc/openvpn/examples/sample-script/bridge-start

```
#!/bin/bash

################################
# Set up Ethernet bridge on Linux
# Requires: bridge-utils
################################

# Define Bridge Interface
br="br0"

# Define list of TAP interfaces to be bridged,
# for example tap="tap0 tap1 tap2".
tap="tap0"
```

```
# Define physical ethernet interface to be bridged
# with TAP interface(s) above.
eth="eth0"
eth_ip="192.168.43.1"
eth_netmask="255.255.255.0"
eth_broadcast="192.168.43.255"

for t in $tap; do
    openvpn --mktun --dev $t
done

brctl addbr $br
brctl addif $br $eth

for t in $tap; do
    brctl addif $br $t
done

for t in $tap; do
    ifconfig $t 0.0.0.0 promisc up
done

ifconfig $eth 0.0.0.0 promisc up

ifconfig $br $eth_ip netmask $eth_netmask broadcast $eth_broadcast
```

A.5 /etc/ufw/before.rules

```
#
# rules.before
#
# Rules that should be run before the ufw command line added rules.
Custom
# rules should be added to one of these chains:
#   ufw-before-input
#   ufw-before-output
#   ufw-before-forward
#

# START OPENVPN RULES
# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
# Allow traffic from OpenVPN client to ens160 (change to the interface
you discovered)
-A POSTROUTING -s 192.168.43.0/8 -o ens160 -j MASQUERADE
COMMIT
# END OPENVPN RULES

# Don't delete these required lines, otherwise there will be errors
*filter
```

```
:ufw-before-input - [0:0]
:ufw-before-output - [0:0]
:ufw-before-forward - [0:0]
:ufw-not-local - [0:0]
# End required lines


# allow all on loopback
-A ufw-before-input -i lo -j ACCEPT
-A ufw-before-output -o lo -j ACCEPT

# quickly process packets for which we already have a connection
-A ufw-before-input -m conntrack --ctstate RELATED,ESTABLISHED -j
ACCEPT
-A ufw-before-output -m conntrack --ctstate RELATED,ESTABLISHED -j
ACCEPT
-A ufw-before-forward -m conntrack --ctstate RELATED,ESTABLISHED -j
ACCEPT

# drop INVALID packets (logs these in loglevel medium and higher)
-A ufw-before-input -m conntrack --ctstate INVALID -j ufw-logging-deny
-A ufw-before-input -m conntrack --ctstate INVALID -j DROP

# ok icmp codes for INPUT
-A ufw-before-input -p icmp --icmp-type destination-unreachable -j
ACCEPT
-A ufw-before-input -p icmp --icmp-type source-quench -j ACCEPT
-A ufw-before-input -p icmp --icmp-type time-exceeded -j ACCEPT
-A ufw-before-input -p icmp --icmp-type parameter-problem -j ACCEPT
-A ufw-before-input -p icmp --icmp-type echo-request -j ACCEPT

# ok icmp code for FORWARD
-A ufw-before-forward -p icmp --icmp-type destination-unreachable -j
ACCEPT
-A ufw-before-forward -p icmp --icmp-type source-quench -j ACCEPT
-A ufw-before-forward -p icmp --icmp-type time-exceeded -j ACCEPT
-A ufw-before-forward -p icmp --icmp-type parameter-problem -j ACCEPT
-A ufw-before-forward -p icmp --icmp-type echo-request -j ACCEPT

# allow dhcp client to work
-A ufw-before-input -p udp --sport 67 --dport 68 -j ACCEPT

#
# ufw-not-local
#
-A ufw-before-input -j ufw-not-local

# if LOCAL, RETURN
-A ufw-not-local -m addrtype --dst-type LOCAL -j RETURN

# if MULTICAST, RETURN
-A ufw-not-local -m addrtype --dst-type MULTICAST -j RETURN
```

```
# if BROADCAST, RETURN
-A ufw-not-local -m addrtype --dst-type BROADCAST -j RETURN

# all other non-local packets are dropped
-A ufw-not-local -m limit --limit 3/min --limit-burst 10 -j ufw-
logging-deny
-A ufw-not-local -j DROP

# allow MULTICAST mDNS for service discovery (be sure the MULTICAST
line above
# is uncommented)
-A ufw-before-input -p udp -d 224.0.0.251 --dport 5353 -j ACCEPT

# allow MULTICAST UPnP for service discovery (be sure the MULTICAST
line above
# is uncommented)
-A ufw-before-input -p udp -d 239.255.255.250 --dport 1900 -j ACCEPT

# don't delete the 'COMMIT' line or these rules won't be processed
COMMIT
```

A.6 client1.opvn

```
################################################
# Sample client-side OpenVPN 2.0 config file #
# for connecting to multi-client server.     #
#                                            #
# This configuration can be used by multiple #
# clients, however each client should have   #
# its own cert and key files.                #
#                                            #
# On Windows, you might want to rename this  #
# file so it has a .ovpn extension           #
################################################

# Specify that we are a client and that we
# will be pulling certain config file directives
# from the server.
client

# Use the same setting as you are using on
# the server.
# On most systems, the VPN will not function
# unless you partially or fully disable
# the firewall for the TUN/TAP interface.
dev tap
;dev tun

# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel
# if you have more than one.  On XP SP2,
```

```
# you may need to disable the firewall
# for the TAP adapter.
;dev-node MyTap

# Are we connecting to a TCP or
# UDP server?  Use the same setting as
# on the server.
;proto tcp
proto udp

# The hostname/IP and port of the server.
# You can have multiple remote entries
# to load balance between the servers.
remote 192.168.42.2 1194
;remote my-server-2 1194

# Choose a random host from the remote
# list for load-balancing.  Otherwise
# try hosts in the order specified.
;remote-random

# Keep trying indefinitely to resolve the
# host name of the OpenVPN server.  Very useful
# on machines which are not permanently connected
# to the internet such as laptops.
resolv-retry infinite

# Most clients don't need to bind to
# a specific local port number.
nobind

# Downgrade privileges after initialization (non-Windows only)
user nobody
group nogroup

# Try to preserve some state across restarts.
persist-key
persist-tun

# If you are connecting through an
# HTTP proxy to reach the actual OpenVPN
# server, put the proxy server/IP and
# port number here.  See the man page
# if your proxy server requires
# authentication.
;http-proxy-retry # retry on connection failures
;http-proxy [proxy server] [proxy port #]

# Wireless networks often produce a lot
# of duplicate packets.  Set this flag
# to silence duplicate packet warnings.
;mute-replay-warnings
```

```
# SSL/TLS parms.
# See the server config file for more
# description.  It's best to use
# a separate .crt/.key file pair
# for each client.  A single ca
# file can be used for all clients.
#ca ca.crt
#cert client.crt
#key client.key

# Verify server certificate by checking that the
# certicate has the correct key usage set.
# This is an important precaution to protect against
# a potential attack discussed here:
#  http://openvpn.net/howto.html#mitm
#
# To use this feature, you will need to generate
# your server certificates with the keyUsage set to
#   digitalSignature, keyEncipherment
# and the extendedKeyUsage to
#   serverAuth
# EasyRSA can do this for you.
remote-cert-tls server

# If a tls-auth key is used on the server
# then every client must also have the key.
;tls-auth ta.key 1

# Select a cryptographic cipher.
# If the cipher option is used on the server
# then you must also specify it here.
cipher AES-256-CBC
auth SHA256

# Enable compression on the VPN link.
# Don't enable this unless it is also
# enabled in the server config file.
;comp-lzo

# Set log file verbosity.
verb 3

# Silence repeating messages
;mute 20

key-direction 1

# script-security 2
# up /etc/openvpn/update-resolve-conf
# down /etc/openvpn/update-resolve-conf
```

A.7 ~/client-configs/make_config.sh

```
#!/bin/bash

# First argument: Client identifier

KEY_DIR=~/openvpn-ca/keys
OUTPUT_DIR=~/client-configs/files
BASE_CONFIG=~/client-configs/base.conf

cat ${BASE_CONFIG} \
    <(echo -e '<ca>') \
    ${KEY_DIR}/ca.crt \
    <(echo -e '</ca>\n<cert>') \
    ${KEY_DIR}/${1}.crt \
    <(echo -e '</cert>\n<key>') \
    ${KEY_DIR}/${1}.key \
    <(echo -e '</key>\n<tls-auth>') \
    ${KEY_DIR}/ta.key \
    <(echo -e '</tls-auth>') \
    > ${OUTPUT_DIR}/${1}.ovpn
```

APPENDIX B. SECURITY ONION /ETC/NETWORK/INTERFACES FILE

```
# This configuration was created by the Security Onion setup script.
#
# The original network interface configuration file was backed up to:
# /etc/network/interfaces.bak.
#
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# loopback network interface
auto lo
iface lo inet loopback

# Management network interface
auto eth0
iface eth0 inet static
  address 192.168.230.105
```

```
  gateway 192.168.230.1
  netmask 255.255.255.0
  dns-nameservers 8.8.8.8 8.8.4.4
  dns-domain csrl-seconion

auto eth1
iface eth1 inet manual
  up ip link set eth1 promisc on arp off up
  down ip link set eth1 promisc off down
  post-up ethtool -G eth1 rx ; for i in rx tx sg tso ufo gso gro lro;
do ethtool -K
  post-up echo 1 > /proc/sys/net/ipv6/conf/eth1/disable_ipv6

auto eth2
iface eth2 inet manual
  up ip link set eth2 promisc on arp off up
  down ip link set eth2 promisc off down
  post-up ethtool -G eth2 rx ; for i in rx tx sg tso ufo gso gro lro;
do ethtool -K
  post-up echo 1 > /proc/sys/net/ipv6/conf/eth2/disable_ipv6
```

```
auto br0

iface br0 inet manual

  bridge_ports eth1 eth2

  up ip link set br0 promisc on arp off up

  down ip link set br0 promisc off down

  post-up ethtool -G br0 rx ; for i in rx tx sg tso ufo gso gro lro;
do ethtool -K

  post-up echo 1 > /proc/sys/net/ipv6/conf/br0/disable_ipv6
```