Sockets

Michael Gamlem III
Computer Science Department
Whitworth University
Spokane, USA
mgamlem19@my.whitworth.edu

Davis Mariotti
Computer Science Department
Whitworth University
Spokane, USA
dmariotti19@my.whitworth.edu

Utsal Shrestha
Computer Science Department
Whitworth University
Spokane, USA
ushrestha20@my.whitworth.edu

***Abstract*—Lorem**
***Index Terms*—Socket, IP Address, Port, Pipelines**

I. Introduction

Modern computer networks rely on many technologies to function efficiently and effectively. The complex system is made of many smaller components each functioning to facilitate communication between two computer endpoints. Among the smaller components is the socket. It is the foundation of the TCP/IP network communication protocol and therefore is the foundation of the modern day internet (Hoffman). However, sockets are known to have security vulnerabilities and no built-in protection against interference or tampering with the information transferred through sockets. Despite this downfall, sockets are able to be used in many different and creative ways including as a communication medium between experimental operating system subsections.

In this paper we explore how sockets are traditionally used, security vulnerabilities that they have, special use cases, and experimental projects utilizing sockets. Additionally, we will explore if sockets are actually the best means of computer communication, particularly in the context of the internet.

II. Definition

A socket is not a physical piece of the computer, but rather, it is the idea that subdivisions can be made on the network interface to separate traffic and route it correctly to where it needs to go. A socket consists of an Internet Protocol (IP) address and a port number. It is commonly written in the form of 192.168.0.1:8080 [6]. Some port numbers, such as 80 and 443, are reserved for specific applications or communication types while others are free for application programmers to utilize. Both IP Addresses and port numbers are "managed" by many local organizations, however the https://www.iana.org/Internet Assigned Numbers Authority (IANA) is commonly regarded as one of the most accurate authority for reference purposes.

III. Types of Sockets

There are a few different types of sockets. They are generally divided into three categories: TCP and UDP socket [17]. TCP (Transmission Control Protocol) is used whenever a more reliable connection is needed, such as when loading a webpage, where missing a few packets might make the entire website look malformed. The second, UDP, is the User Datagram Protocol. This is an alternative protocol used when it is not as important to ensure all data is transmitted accurately. A common use case for this protocol would be video/audio streaming, where it is more important to transmit data faster at the cost of missing a few frames.

A. TCP

TCP is the more commonly used in the Internet than UDP. This is because the protocol is centered around reliability, and accurate transmission of data. The way that TCP achieves this is by numbering each packet so that the client receiving the data is able to sequence them correctly [**?**]. After receiving each packet, the client sends a message to the server confirming that it received the packet. If the server does not receive this confirmation, it can resend the packets to ensure there was an accurate transmission of data. TCP thus is used more the

majority of applications as accurate transmission of data is generally regarded as of utmost importance.

B. UDP

UDP works similarly to TCP, except it does no error checking or packet numbering (Hoffman). The main advantage is that it is much faster than TCP. The disadvantage is that some packets could be lost and there won't be a way to correct the lost data. In a client server relationship, the client requests some data, and the server sends all the packets back at once without stopping. This can be advantageous when error checking isn't necessary. One example is video streaming. If the connection is lost momentarily, there may be a loss of video or it may get glitchy, but it is more important to continue playing the video as is than to ensure it was all received correctly. With TCP, there may be too much latency to watch a live stream, whereas it would work well with UDP.

IV. Uses

A. General Programming

Sockets have a wide array of uses when doing general programming. For example, programs such as Telnet, FTP, and the World Wide Web use sockets (Mitchell). Sockets can be used for interprocess communication on a single computer, or they can be used over networks to transfer data bidirectionally. Generally, network programming uses the client server model. This means that there is a client requesting data, and a server responding with data. In order to do this, the client must know the IP address and port that the socket has been created at [17]. On the server side, sockets can be used to handle requests coming from many different sources at the same time. Either the process that created the socket can handle requests iteratively, processing each one by one, or concurrently, using fork to create a child process to handle the request.

B. Interprocess Communication

Sockets can be used for interprocess communication where all data transfer occurs on the same machine. This can be useful in instances where you want to separate different concerns of an application into microservices, which has become a very popular in the last decade. To accomplish this, the developer would likely expose each microservice as a socket (likely a websocket) that other services and processes communicate to them with.

V. Security

As with most aspects of the internet today, security is an issue that cannot be ignored when discussing web sockets. Even though the standard has been around for decades, it has not been reliably or adequately updated to protect against modern security threats. This is due to many complicating factors that are outside the scope of this paper. However, below are some examples of both commonly exploited and uncommonly exploited vulnerabilities in the socket communication protocol as well as potential experimental solutions.

A. Vulnerabilities

1) No Built-In Security Measures: There is no security built into the socket protocol. This means that all applications that use sockets must be designed to protect against any and all possible attack vectors. Some protections are given by host operating systems; however, they are not all-inclusive and not always sufficient. This results in inconsistent security practices among applications who utilize sockets and an inability to quickly fix a security problem as many organizations must act separately to solve any security concerns.

At the Black Hat conference of 2009 a security researcher gave a presentation where they showed how simple it was to intercept Secure Socket Layer (SSL) traffic and compromise a user's secret login data by performing a man-in-the-middle style attack [18]. This traffic would be traveling across a socket from the user's machine to the login server. Due to the fact that information traveling over sockets is not encrypted or otherwise protected by default. This researcher was able capture essentially all data that users were entering to the compromised website.

The attack was only able to be carried out when users were moving from an unsecured http page to a secured https page to log into the compromised service. The researcher was then able to intercept and reroute traffic to their proxy server before directing the traffic to its original destination [18]. If the socket protocol included ways to verify that traffic had not been intercepted on its way to a

destination or encrypted all information sent over a socket by default, this attack would not be able to be carried out.

2) Blind Trust: Due to the fact that operating systems protect I/O devices from being directly accessed by applications, the applications must trust that the host operating system is delivering data exactly as it appeared over the socket [16]. This is typically not an issue, however in cases where the host operating system has been taken over by a malicious entity or otherwise compromised, problems quickly arise.

Traditional computers and operating systems have no workable solution to this problem. Therefore, a new type of computer operating system called Proxos was developed by researchers at the University of Toronto with the specific goal of addressing this blind trust problem.

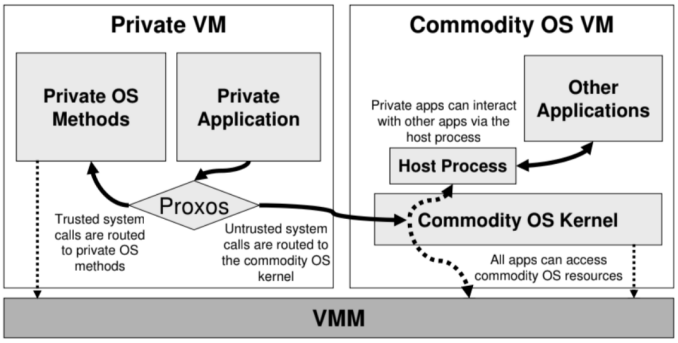

Fig. 1. Diagram of the Proxos System Architecture [16]

As can be seen in the above figure, this system architecture is made up of three or more Virtual Machines (VM). The Virtual Machine Master (VMM) would be responsible for overseeing all other virtual machines that operate on the system. It would also be responsible for allocating resources to each VM as they are created or destroyed [16]. It is not a traditional operating system. It is only intended to manage hardware resources and respond to requests made to those resources. The Commodity Operating System (OS), the researchers used Linux, would be what we are used to today, the traditional user operating system. It would be allowed to run application as any other normal operating system would. The only difference is that it is not responsible for handling hardware resources and can only ask the VMM for the resources it needs [16].

The main difference between Proxos and traditional computer systems lies in how it handles applications that want to have more security. These applications are given their own private VM with their own copy of the OS that can make trusted calls directly to the VMM and bypass a potentially compromised host operating system. Only the application is allowed in this private VM. Its application-trusted system calls are allowed to communicate directly with the VMM to get information directly from hardware resources while untrusted system calls are sent back to the Commodity OS [16]. This means that, regardless of if the host operating system that the user is running can be trusted, programs running in this private VM environment can be sure that there are no security risks in their application or operating system scope as there is no way for the VMM to be compromised.

According to the paper published by the researchers this enormous gain in security came with only minimal performance reduction when compared to running identical applications on Linux. Furthermore, existing applications were able to be modified to work with this system with modifying very little code (as few as 10s of lines).

3) Distributed Denial of Service Attacks: Distributed Denial of Service (DDoS) attacks are simple in purpose and in execution. They are designed to disrupt communication over a network. They do so by using multiple hosts to disrupt communication by overloading a socket with bogus information requests or other meaningless traffic [7].

Unfortunately, sockets are designed to take and process all information that is assigned so they, and therefore the targeted system alone, are defenseless against this style of attack. To further complicate the detection and prevention of DDoS attacks, the attacking host(s) may inaccurately report (spoof) the socket that the traffic originates from, therefore the targeted system is unable to determine which requests are legitimate and which requests are from an attacking system [7].

B. Potential Solutions

Many companies and researchers have come up with solutions to address the security vulnerabilities of sockets. Some of these solutions have been widely adopted, such as rate limiting, while others,

such as artificial intelligence (AI) or neural network based approaches, are beginning to be adopted by the industry and end users. What follows is a summary of some ways the many security vulnerabilities of sockets can be addressed.

1) Traffic Filtering: Many network routing and switching appliances come with software solutions to protect against common socket exploits. Most commonly, these network appliances limit the number of requests that that can be made from or to clients in order to ensure fair network use amongst all clients [7]. While the real intent is to provide fair network access, as a side effect some protection is given to against DDoS attacks. This is due to the fact that sockets are less likely to be overwhelmed by excessive traffic as the flow rate of traffic is modified as the rate of requests from one or to one source is limited.

2) Smart Wire: This is a concept where a monitoring device is inserted on a normal communication wire and passively monitors the traffic that passes across the wire [7]. When the monitoring device detects a threat of any sort, it can update routers, firewalls, and even antivirus software that a threat is being transferred and pinpoint its source and destination sockets. This would allow systems on the network to be warned about the attack and take any appropriate defensive measures (such as blocking traffic from the originating socket).

This technique only works when the monitoring device is able to monitor traffic without creating a bottleneck and thereby reducing the throughput of the smart wire. Additionally, precautions must be taken to ensure that the monitoring device is not compromised. Because it sits between source and destination sockets, it is in the perfect place to execute a man-in-the-middle attack such as redirecting traffic to another destination or stealing private information that may be transferred across the smart wire.

3) Agent Based Threat Detection: Another common way to detect threats is to use agents. In this setup each network device would have a copy of the base model used to determine if traffic (or any other activity) is a threat. The threat detection artificially intelligent agents would periodically scan to determine if any suspicious activity is occurring and respond as necessary.

There are many benefits to using agents as threat detection tools. One is that there may be multiple agents working to accomplish the same task or each may be specialized to perform a subset of the larger task [1]. One example of this could be one agent being used to detect active network threats coming across a socket such as DDoS attacks while another is looking for more passive threats such as port scans. This allows information to be processed more quickly and at a more granular level as opposed to a non-agent based approach [1]. Another benefit is the ability for a network of many deployed agents to learn independently as they differing types of attacks and update each other with the knowledge they gain detecting attacks. This is the approach that is taken by many modern-day antivirus applications [1].

VI. Experimental Uses

There are many ways that researchers are attempting to adapt or improve the traditional idea of a socket. Because the socket is so widely used and is not owned by any particular organization or individual adaptations and improvements are essentially risk free and have the potential to impact a wide variety of users. What follows is a summary of one innovative idea and how it may impact the future of network based communication

A. MuniSocket

This is an experimental middle-layer API that can allow one socket to be used on multiple network interfaces [13]. The problem that it can solve is the need for large data transfers to be done more rapidly. As of now, most network transfers come from one source socket and go to one source socket. Due to the limitation that a socket can only be active on one network device at a time, this significantly limits the speed at which data can be transferred.

As you can see in Figure 2 the MuniSocket allows for Applications to act as if they are interfacing with one socket directly and then subdivides all traffic to be transferred over a socket to one or more network interfaces. This results in three main benefits. The speed of the transfer is increased due to more throughput potential, it becomes possible to balance the load of network cards, and fault tolerance becomes more feasible as the MuniSocket

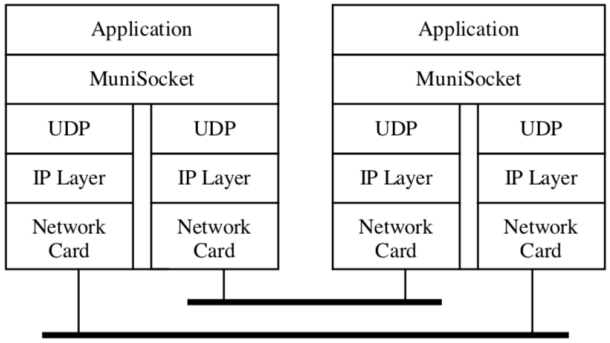

Fig. 2. Diagram of the UDP version of a MuniSocket [13]

is able to re-route the traffic without the application knowing that a fault has occurred [13].

The MuniSocket can operate in both a UDP and TCP configuration [13]. This is due to the fact that the MuniSocket still uses the traditional socket protocols to transfer data. Both the reliable TCP and unreliable UDP MuniSockets saw similar performance in terms of data transfer speed to their traditional counterparts. Both the TCP and UDP configurations of the MuniSocket were found to have a higher throughput capacity than their respective traditional counterpart (see 3 and 4) [13]. The paper offered no explanation as to why this may be but it is reasonable to assume that this is due to the increased throughput capacity of having multiple network interfaces.

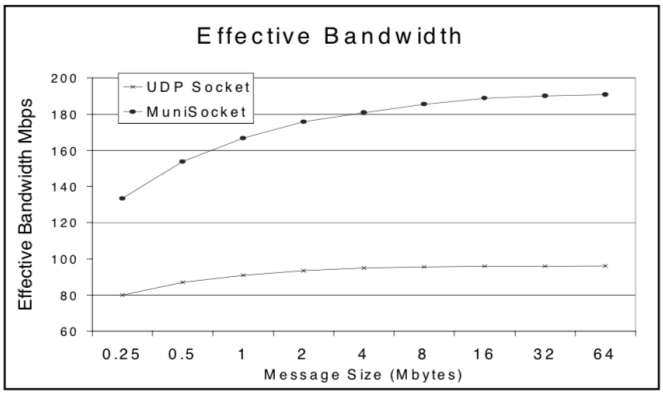

Fig. 3. Results of testing UDP MuniSocket bandwidth [13]

Although this technology seems promising and is an innovative way to interface with sockets testing revealed that only moderate performance gains were possible with the current implementation. Figure

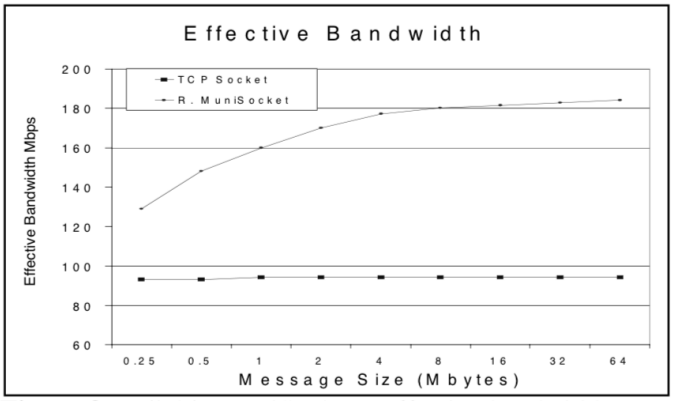

Fig. 4. Results of testing TCP MuniSocket bandwidth [13]

5 shows that in the tests performed only slight increases in transfer speeds were observed when using multiple interfaces to balance the load of the transfer. It is possible that this was due to the artificial nature of the test and real-world scenarios would reveal better results. Nevertheless, the minimal performance gains do not inspire hope that the MuniSocket will be widely adopted.

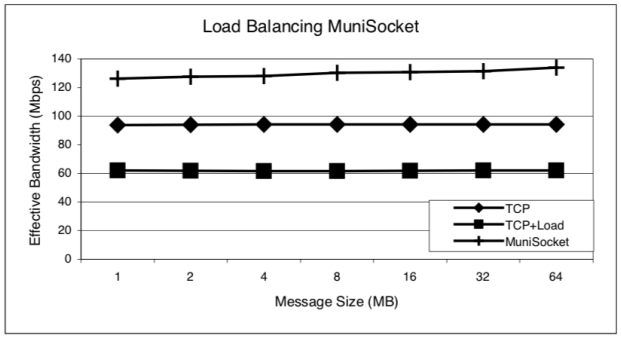

Fig. 5. Results of load balancing test for MuniSocket [13]

VII. Comparison

A. Remote Procedure Calls

Remote Procedure Call (RPC) is a protocol that can be used to request a service from a program located in another node on a network without having to understand the network's details. It is also called a function call or a subroutine call (Rouse 2019). It uses the client-server model and is a synchronous operation requiring the requesting program to be suspended until the results of the remote procedure

are returned. Sockets are a way of implementing remote procedure call services, and are quite low level compared to RPCs, which hide the communication details behind procedural calls. The use of RPC compilers allows for the communication protocol to be generated automatically from the interface description.

REFERENCES

[1] Arabo, Abdullahi. Distributed IDS using Agents: An Agent-Based Detection System to Detect Passive and Active Threats to a Network. Reading: Academic Conferences International Limited, 2019. ProQuest. Web. 10 Apr. 2019.

[2] Bailey, Kimberly Tekavec. AF UNIX Socket Across Systems in the Same Computer on Computer Systems that Support Multiple Operating System Images. International Business Machines Corp, assignee. Patent US20080155103A1. 2008.

[3] de, la C., et al. "Checking the Reliability of Socket Based Communication Software." International Journal on Software Tools for Technology Transfer 11.5 (2009): 359-74. ProQuest. Web. 10 Apr. 2019.

[4] Ganger, Gregory R., et al. "Fast and Flexible Application-Level Networking on Exokernel Systems." ACM Transactions on Computer Systems 20.1 (2002): 49. ProQuest. Web. 10 Apr. 2019.

[5] Gay, Warren W. "Linux Socket Programming: By Example". Que Corp. Indianapolis, IN, USA, 2000.

[6] Goralski, Walter. The Illustrated Network : How TCP/IP Works in a Modern Network, Elsevier Science & Technology, 2008. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/whitworth/detail.action?docID=405967.

[7] Herbert, Alan, and Barry Irwin. "DDoS Attack Mitigation Through Control of Inherent Charge Decay of Memory Implementations." Iccws 2015-The Proceedings of the 10th International Conference on Cyber Warfare and Security: ICCWS2015. Academic Conferences Limited, 2015.

[8] Kalita, Limi. "Socket Programming". International Journal of Computer Science and Information Technologies, Vol. 5 (3), 2014, 4802-4807.

[9] Law, KL Eddie, and Roy Leung. "A design and implementation of active network socket programming." Microprocessors and Microsystems 27.5-6 (2003): 277-284.

[10] Leffler, Samuel J., et al. "An Advanced 4.4BSD Interprocess Communication Tutorial". University of California, Berkeley. 1986.

[11] Matthew, Neil, and Richard Stones. Beginning linux programming. John Wiley & Sons, 2008.

[12] Mitchell, Mark, Jeffrey Oldham, and Alex Samuel. Advanced linux programming. New Riders Publishing, 2001.

[13] Mohamed, Nader, et al. "A USER-LEVEL SOCKET LAYER OVER MULTIPLE PHYSICAL NETWORK INTERFACES." ProQuest, University of Nebraska – Lincoln, middleware-tech.net/papers/PDCS2002_IASTED_MuniSocket.pdf.

[14] Oliver, Stephen L., et al. "Scheduling task parallelism on multi-socket multicore systems". Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers. Tucson, Arizona, 2011, 49-56.

[15] Reed, Dennis F., et al. "Interprocess communications system and method utilizing shared memory for message transfer and datagram sockets for message control". US5652885A, United States Patent and Trademark Office. 1993.

[16] Richard Ta-Min, Lionel Litty, and David Lie. 2006. Splitting interfaces: making trust between applications and operating systems configurable. In Proceedings of the 7th symposium on Operating systems design and implementation (OSDI '06). USENIX Association, Berkeley, CA, USA, 279-292.

[17] Sawant, Abhijit A., and B. B. Meshram. "Network programming in Java using Socket." Network 3.1 (2013).

[18] Sheble, Nicholas. "Black Hat Conference: Socket to Me." Intech 56.4 (2009): 9. ProQuest. Web. 10 Apr. 2019.

[19] Stevens, W. Richard, and Stephen A. Rago. Advanced programming in the UNIX environment. Addison-Wesley, 2008.

[20] Xue, Ming and Changjun Zhu. "The Socket Programming and Software Design for Communication Based on Client/Server". 2009 Pacific-Asia Conference on Circuits, Communications and Systems. IEEE. 2009.