WiFi Traffic Forwarding Client and Server

William tenHaaf
Grand Valley State University

WiFi Traffic Forwarding Client and Server

By
William ten Haaf
April, 2020

WiFi Traffic Forwarding Client and Server

By
William ten Haaf

A project submitted in partial fulfillment of the requirements for the degree of
Master of Science in
Computer Information Systems

at
Grand Valley State University
April, 2020

Dr. Greg Wolffe **Date**

Table of Contents

Abstract

WiFi traffic is ubiquitous. Between video game consoles, portable computers, smart phones, and even some desktop computers, the vast majority of consumer electronics have WiFi capabilities. Occasionally, these devices require direct, ad-hoc WiFi connections in such a way that communication over the Internet, or even a single hop, is impossible. In such cases the only options for increasing communication range are to use WiFi repeaters or to upgrade device antennas. Depending on the device, this can be difficult or impossible to do. These limitations create a need for the ability to expand a WiFi network in a data-agnostic way that supports a longer range than repeaters or antenna upgrades allow.

To meet this need, we developed a set of programs that capture WiFi traffic and relay it to a remote location. The programs are designed to allow devices in remote locations to communicate with each other without the use of proxies, VPNs, or any technologies that require special device configurations. Because raw WiFi frames are captured and retransmitted, devices which require direct WiFi connections have their traffic rerouted to a remote location without any configuration requirements on the local device and the entire process is completely transparent to the devices that utilize it. To prove the viability of the system, we tested communication via several different applications. We also offer some other potential applications and suggestions for future work.

Introduction

This project implements a method of transferring raw WiFi traffic from one location to another. This program suite is intended to allow devices to access a remote WiFi network or WiFi devices transparently and without any specific device configuration requirements. Certain devices, such as video game consoles and industrial hardware, have limited capabilities regarding wireless configuration, which can render them unable to communicate wirelessly across IP networks for certain tasks. This project aims to provide a solution to that problem and to other issues.

Background and Related Work

Currently there are no existing solutions to solve the previously mentioned problems. Related solutions require proxy or VPN capability on client devices, which does not help to solve the specified issues. This project works transparently and does not require any special device configuration besides standard WiFi communication to function. Wireless repeaters provide some similar functionality, but most consumer repeaters do not appear to work with ad-hoc wireless traffic and repeaters in general are not suitable for Very Long Distance communication.

Project Details

Specification

The proposed solution transfers WiFi traffic from one location to another, and vice versa, allowing distributed WiFi devices to communicate with each other as if they were in the same room. Due to its implementation protocol, it also allows functionality such as traffic generation and traffic analysis.

Implementation

The project uses `libpcap` to capture WiFi traffic from the wireless NIC and TCP sockets for client/server communication. Knowledge of TCP networking was essential, as was a familiarity with tools such as Wireshark. The capture functionality required additional research in support of development.

For client-server communication, WiFi traffic is simply a payload that the client generates and processes, prepended by the size of the payload. The array that contains the payload size information must have the same endianness as the server, therefore, clients must be implemented to account for that. The payload can be arbitrary data, but the clients must all be able to parse or ignore it if mixed-purpose clients are utilizing one server. For this project, the clients only processed compressed WiFi traffic.

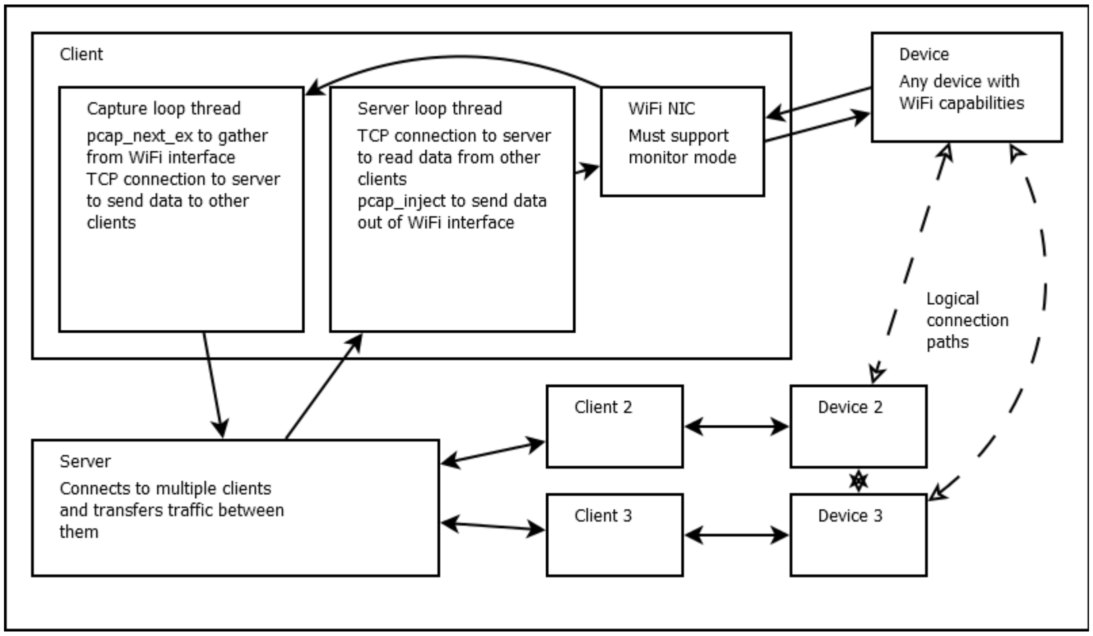
Client
Capture loop thread
pcap_next_ex to gather from WiFi interface
TCP connection to server to send data to other clients
Server loop thread
TCP connection to server to read data from other clients
pcap_inject to send data out of WiFi interface
WiFi NIC
Must support monitor mode
Device
Any device with WiFi capabilities
Logical connection paths
Server
Connects to multiple clients and transfers traffic between them
Client 2
Device 2
Client 3
Device 3

Figure 1: A graphical representation of the client data flow

The client is responsible for capturing the traffic, sending the traffic to the server, and broadcasting the traffic. The data flow process for the client is shown in Figure 1. The client uses a thread for capturing traffic and uses the libpcap function **`pcap_next_ex()`** to read wireless frames from the monitor interface. Once the frame is read, it is compressed using the Zstandard compression algorithm and sent to the server over TCP. When the server sends a client a frame, the client decompresses it with the Zstandard compression algorithm. After decompression, the frame is sent to the wireless interface with libpcap's **`pcap_inject()`** function and broadcast over the local wireless channel. Because traffic is not modified by the clients or server, the traffic that is seen by a local device is the same traffic that was sent by a remote device, which allows local and remote devices to communicate as if they were within wireless range of each other.

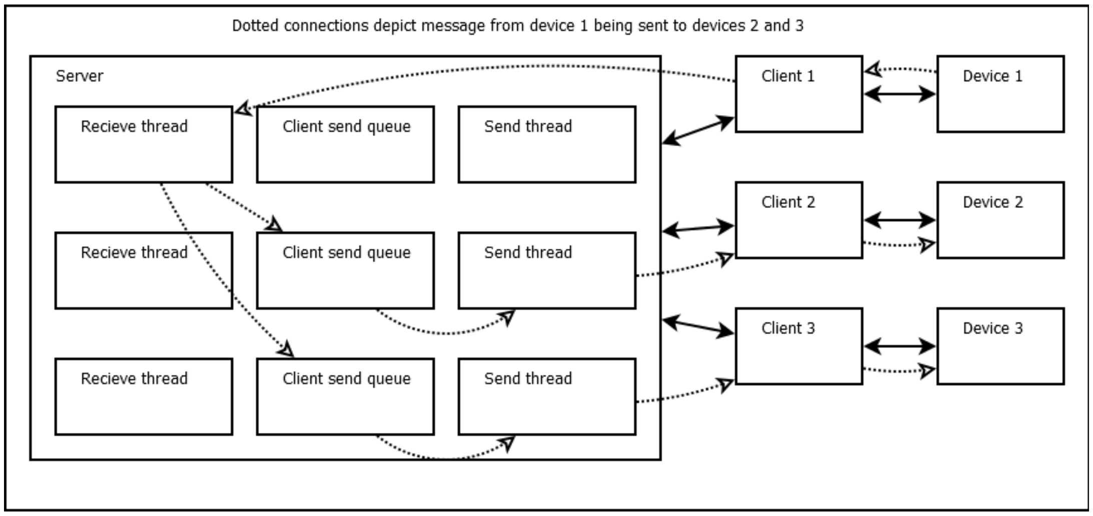

Figure 2: A graphical representation of the server data flow

The server is responsible for receiving traffic from the clients and sending received traffic to every other client, as detailed in Figure 2. The server creates a TCP receive thread and a TCP send thread for each client that connects. Additionally, the server keeps packet send queues for each client. The server receives traffic from a client and adds it to each connected client's send queue and signals a mutex for the respective client send thread. Upon being signaled, the send thread will read packets from the queue and send them to the client until the queue is empty, after which it will wait for new data to be added to the queue. Because no processing is done on the received data, the server can be repurposed to work with any clients that follow the traffic pattern.

Results, Evaluation, Reflection

Under various testing scenarios, all standard WiFi traffic was compatible with this system. As specified, the system's functionality was transparent to the devices using it and no device configuration was required. Performance on typical client/server infrastructure meets specifications, and the system has source-configurable compression settings that allow further optimization of bandwidth or CPU usage. Because this prototype prioritized performance, data encryption was not employed; systems should use SSH or some other encapsulation technique if secure communication is desired.

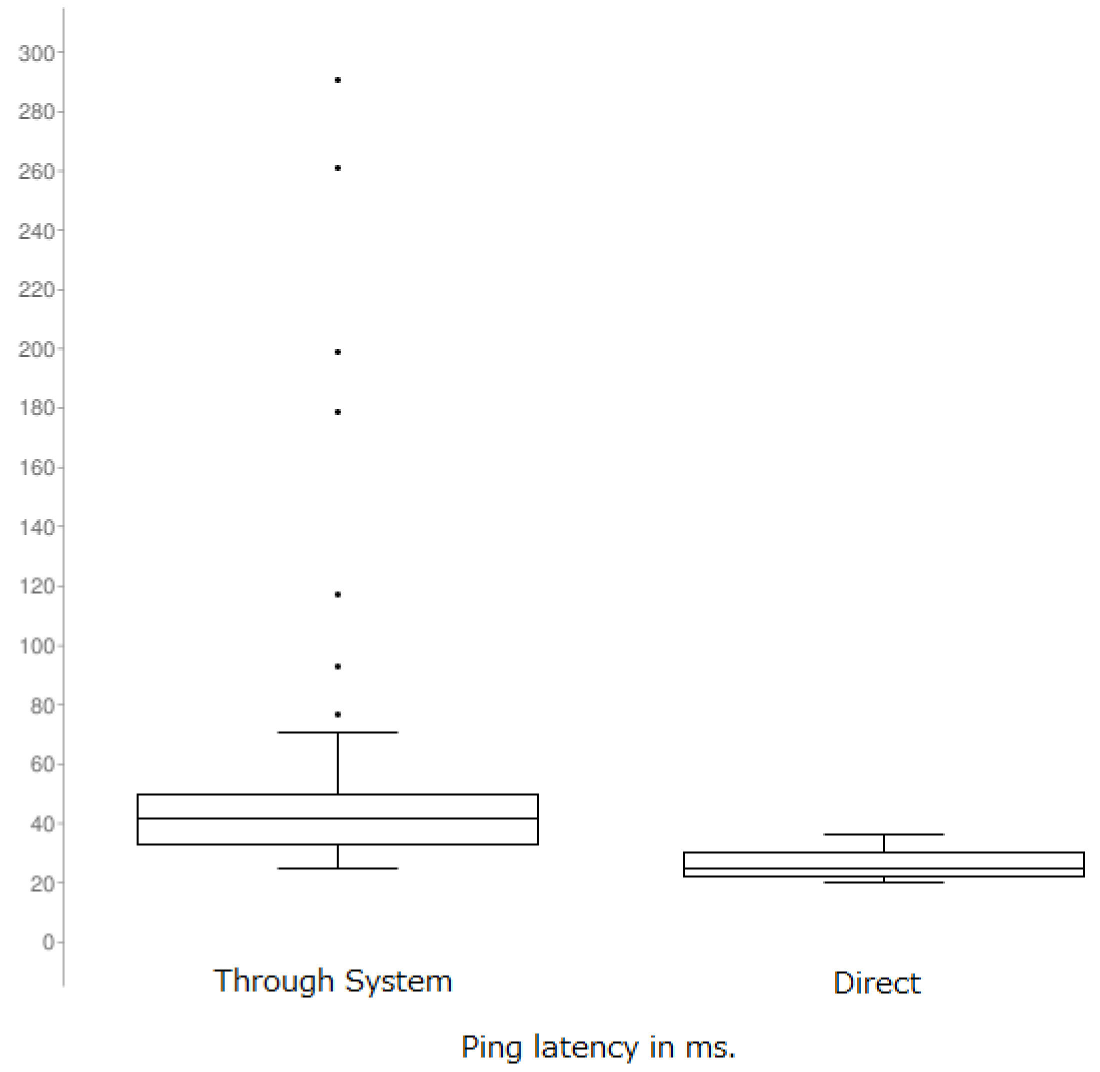

Figure 3: Latency comparison with and without the forwarding system

Due to the addition of the client/server infrastructure, latency between devices using the system will be increased by at least the latency between the clients and the server. On high performance networks and hardware, this can be several milliseconds, which, depending on the application, may be acceptable. Figure 3 displays latency data observed using standard consumer hardware on a busy network. Data collection hardware and methods are described in the Appendix. For the collected data, average latency was around 15 milliseconds and the minimum latency difference was around 3 milliseconds. The system shows outliers that were generated when heavy traffic usage occurred on one of the networks. Higher capacity or less busy networks would be expected to exhibit a more predictable latency increase.

There are some interesting potential applications that were not tested in the course of this research. For example, powering displays by transferring Miracast/WiFi Direct traffic over the Internet could be utilized when running Ethernet is prohibitively expensive. Depending on the hardware used, it may not be possible to capture the large frames that WiFi Direct traffic can produce, though it should be possible to overcome those issues by using modern wireless adapters.

The system is configured with a “star” network topology, so the amount of traffic scales linearly with the number of connected clients. For large numbers of clients and high bandwidth applications, the server can become a bottleneck.

Currently, client programs do not employ error recovery and return to the shell when disconnected from the server. Reliability is achieved by calling the client in a Bash loop, but it would be slightly more efficient to have that functionality built into the client. There would not be a noticeable difference in functionality or performance unless there was a very unstable connection between the client and server.

Conclusion and Future Work

The system successfully implements a functionality that allows remote WiFi devices to communicate with each other as if they were completely local. Future work includes testing for edge cases like large WiFi packets from Miracast and similar technologies. To allow for more portable code, endian-agnostic payload size code should replace the current integer conversion code. Investigations into whether or not a mesh topology or TCP multicast can resolve performance bottlenecks with large numbers of clients is also another area of research. Client/server communication encryption is a desirable feature, allowing for plain text WiFi data to be transferred over the Internet or other insecure medium.

Bibliography

Libraries

Zstandard: https://github.com/facebook/zstd. This library was used to compress the data that the clients send to each other.

libpcap: https://www.tcpdump.org/. This library was used to read data from and inject data into the wireless interface.

Tools

Wireshark: https://www.wireshark.org/. This tool was used to build WiFi access point test beacons and to debug WiFi traffic injection and WiFi traffic capture.

Aircrack-ng: https://www.aircrack-ng.org/. This tool was used to put the wireless interface into monitor mode and to debug WiFi traffic injection.

References and APIs

Zstandard: https://facebook.github.io/zstd/zstd_manual.html.

libpcap: https://www.tcpdump.org/manpages/pcap.3pcap.html.

Project Source Code

WiFi Traffic Forwarding Client and Server:

https://github.com/william-G2hOux9g8q/WiFiTrafficForwardingClientandServer.

This is the public source code repository for the programs in the project.

Appendix

The data in Figure 3 was obtained by creating an ad-hoc connection between two laptops through the system described in this report. The hardware used to run the server was a Hyper-V virtual machine. Two ARM Cortex-A53 single board computers were used for the system clients, and two Dell Latitude E6430 laptops were used for the test WiFi devices. The test was performed over an Internet link with the remote client on an AT&T 1.5 Mbps DSL connection and the server on an 80 Mbps Comcast cable connection. The data for the direct connection was obtained by creating a direct ad-hoc connection between the Dell laptops. For generating the data points, simple ICMP pings were used and the response time was recorded.