Securing the Web with Rust: A Case Study on Acoustic Networks

Linshu Yang
Shanghaitech University

Lei Huang
Shanghaitech University

Abstract

Modern society's heavy reliance on software automation demands trustworthy and secure code. Despite rigorous testing, memory-related vulnerabilities persist in languages like C and C++. This study explores Rust, developed by Mozilla, as a secure alternative. Rust's ownership model and borrow-checking rules offer a zero-cost solution, ensuring both high performance and memory safety. We present Rathernet, a high-performance network stack implemented in Rust for acoustic channels, supporting OSI model protocols and Internet Protocol. Rathernet incorporates Acoustic Link, Multiple Access management, internet connectivity, and extends above the IP layer to support DNS and HTTP. This work showcases Rust's effectiveness in enhancing memory safety, providing a secure foundation for real-world applications.

1 Introduction

Modern society relies heavily on software-based automation, implicitly trusting developers to write software that operates in the expected way and cannot be compromised for malicious purposes. While developers often perform rigorous testing to prepare the logic in software for surprising conditions, exploitable software vulnerabilities are still frequently based on memory issues.

Commonly used languages, such as C and C++, provide a lot of freedom and flexibility in memory management while relying heavily on the programmer to perform the needed checks on memory references. Simple mistakes can lead to exploitable memory-based vulnerabilities. Software analysis tools can detect many instances of memory management issues and operating environment options can also provide some protection, but inherent protections offered by memory safe software languages can prevent or mitigate most memory management issues.

The Rust programming language [5], developed by Mozilla, stands out as a modern and secure alternative to C and C++, addressing the enduring challenge of memory safety in software development. Rust's distinctive ownership model and borrow-checking rules provide a zero-cost solution, allowing developers to write high-performance code while ensuring safety. This emphasis on preventing memory-related pitfalls makes Rust an attractive choice for projects prioritizing security and reliability.

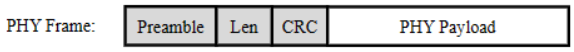

Figure 1: Physical Layer Frame Structure

In this work, we implemented a high performance network stack running on acoustic channels with Rust. The network stack, named as Rathernet, reproduces many features and protocols from different layers of the OSI model [7]. It also supports the standard Internet Protocol [1] and is able to be integrated into the broader internet infrastructure, accommodating the communication of higher layer applications.

2 Acoustic Link

2.1 Modulation and Framing

In the Rathernet's physical layer, we implemented PSK modulation for every bit in the payload. This involved encoding each bit onto a 5k Hz sound wave during air transmission, with 0 corresponding to a phase of 0 and 1 to a phase of π.

$$Zero = \sin 2\pi ft$$
$$One = \sin(2\pi ft + \pi)$$

Prior to payload transmission, a warm-up sequence of 8 bits, all set to 1, is played to prepare the speaker. The payload itself is encapsulated within a 64-bit packet, featuring a chirp preamble and header of 32 bits and 4 bits, respectively, and a checksum of 16 bits. The packet is terminated by a countdown, with the header encoding the packet's length.

2.2 Error Detection and Correction

To enhance the dependability of our acoustic link, we employ the Reed-Solomon encoding [10] method to encode the payload. This method involves dividing the data into shards, and in our implementation, we represent each packet as an individual shard containing 8 bytes. To maintain uniformity, any remaining bytes are padded with zeros. The actual valid length of the packet is stored in the header, ensuring accurate reconstruction during decoding. To minimize computational costs for our specific use case, we opt for CRC16 IBM [11] over CRC32 to calculate the checksum. This choice is rooted in the belief that CRC32 proves too resource-intensive for our application. Ultimately, the payload undergoes decoding as a complete entity once all shards are received, and the integrity is confirmed through verification using the CRC. This comprehensive approach ensures a reliable and efficient acoustic link for our system.

In response to the suboptimal performance of our speaker, we implement an additional layer of encoding for the packet header using convolutional code [12] with a specific configuration of (7, [171, 133]). The convolutional code provides error correction capabilities, particularly beneficial for mitigating issues arising from the speaker's limitations. Upon reception, the encoded header undergoes decoding through the Viterbi algorithm [1], ensuring the accurate retrieval of information. While this step may not be imperative under optimal speaker conditions, convolutional code serves as an supplementary safeguard. Recognizing the subpar performance of convolutional codes on short sequences, its inclusion in our protocol is a strategic measure to bolster the robustness of our communication system, especially in instances where the speaker's functionality may be compromised.

2.3 Higher Bandwidth

Considering the quasi-linear nature of sound waves, we have chosen to enhance our approach by integrating Orthogonal Frequency Division Multiplexing (OFDM) [8]. This involves combining multiple orthogonal signals at distinct frequencies to optimize the overall system for improved recognition and efficiency.

By experiment, we make use of DFT to filter out the carriers' frequencies instead of using symbols' inner product directly.

$$Zero_k = \sin 2\pi(f + k\Delta)t$$
$$One_k = \sin(2\pi(f + k\Delta)t + \pi)$$

where $\Delta = 1kHz$, since the DFT has the resolution of $1kHz$ given the bit rate at the same frequency.

2.4 Range Challenge

To enhance data transmission over extended distances, our goal is to address the increased complexity of sound waves as they propagate, presenting challenges like multipath effects. This task focuses on motivating groups to achieve longer transmission distances. To extend the range, we replaced our 5kHz symbol-encoded sine wave with two sweeping sounds at different frequency ranges. Drawing from our knowledge, we recognize the rarity of sweeping signals in nature, ensuring robustness in transmission. Subsequently, our system can reliably transmit at distances of no less than 2 meters.

3 Manage Multiple Access

3.1 CSMA/CD

This segment is implemented based on the WARP CSMA-MAC [4] . The CSMA protocol [6] is implemented as follows:

Conceptually, three threads operate in the system:

1. MAC Threads: Accepting data from the source, providing instructions to the Write Thread, and receiving data from the Listen Thread.

2. Listen Thread: Continuously monitoring the channel, sending data to the MAC.

3. Write Thread: Awaiting data and instructions from the MAC, deciding to send when the channel is clear.

Additionally, a back-off timer is set after a clearing timeout, although this is not explicitly illustrated in the automaton. This prevents the CSMA/CD automaton from continuously accepting packets from the source, avoiding potential issues.

A more elucidating figure is presented in 2.

3.2 Selection of Parameters

Notably, from here our transmission medium is changed to 3.5mm wire instead of the air. Before tuning the CSMA parameters, it is pertinent to highlight several key parameters:

1. **Bit Rate:** $15kHz$, as it no longer requires the production of mechanical waves.

2. **Slot Length:** $85ms$. This empirical value is crucial in the CSMA system for detecting a busy channel and initiating random wait times. The slot length should ideally exceed the propagation delay slightly and may need adjustment if noise interference persists.

3. **ACK Timeout:** $30ms$. This duration represents the measured delay from the successful end of frame transmission to the ACK receipt event.

Figure 2: CSMA

To optimize performance, careful parameter selection is paramount. Through numerous experiments, we have identified the following aspects as most impactful on the final outcome:

1. Restrict the length of random backoff.

2. Set the ACK timeout as close to the ground truth as possible.

3. If channel detection relies on monitoring energy levels,

With appropriately tuned parameters, we attained transmission completion in less than 30s for the designated task, even in noisy conditions.

3.3 Avoidance for the Abuse of Channel

There exists such a situation: after a collision, the CSMA system decides the nodes to backoff for t_1 and t_2 seconds. WLOG $t_1 < t_2$ and denote the frame lenght t_0 seconds. Suppose an ACK is sent from node 2, and it is received at t' where $t_1 < t' < t_2 \wedge t_2 - t' < t_0$. So at this moment, node 1 will restart immediately and send new frames. And it blocks the node 2 since $t_2 - t' < t_0$ (node 1 is playing the frame when the CSMA of node 2 ends its backkoff). All in all, node 1 will continues sending process smoothly, but node 2 is likely to backoff until node 1 ends all transmission.

This can be coped with a tight parameter settings, and rarely happens in the Rathernet.

4 To the Internet

4.1 Router and NAT

The router seamlessly manages communication between the Rathernet and the internet in both inbound and outbound directions. To facilitate this interaction, we use the PCAP library [**?**] to capture incoming packets from the internet. By extracting information from the IP packet header, the router identifies the destination of the packet and forwards it to the Rathernet if necessary. This process lays the groundwork for implementing Network Address Translation (NAT), which can be further enhanced with dynamic routes learned from outbound traffic and static routes predetermined through configuration settings.

In contrast to TCP and UDP, which rely on ports for connection identification in NAT, ICMP lacks a port attribute. Nevertheless, we address this limitation by utilizing the ICMP Identifier as a workaround. This alternative approach allows the router to effectively manage and track ICMP connections within the NAT environment, ensuring a comprehensive and seamless network communication experience.

4.2 Virtual Network Device

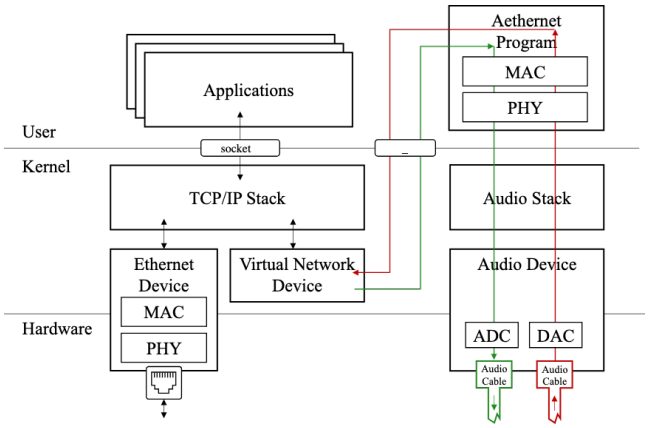

Figure 3: The TAP TUN structure

TUN technology enables network tunneling by simulating network layer devices, allowing packet transmission and reception as if they were raw network layer protocols, critical for implementing virtual network interfaces in VPNs and other networking solutions.

WinTun [**?**] is a minimal, high-performance tunneling library for Windows, enabling the creation of fast and efficient network tunnel interfaces, vital for VPN and other networking applications.

The Rust WinTun library is a Rust binding for the WinTun library, offering efficient and easy-to-use interfaces for tunneling and networking in Rust applications on Windows.

It leverages WinTun's high-performance capabilities, providing Rust developers with a powerful tool for building virtual network device solutions.

Accompanied with the PCAP library, ICMP Echo and other application level protocols can work flawlessly on the Rathernet.

4.3 Address Resolution Protocol

In the Rathernet, the Address Resolution Protocol (ARP) [2] operates at the same level as IP. However, to streamline its functionality within local networks, we opted for simplification. Our design involves crafting ARP as a distinctive MAC packet, encapsulating both source and destination IP addresses. This innovative approach not only simplifies the ARP process in the Rathernet but also ensures the preservation of its essential functionality.

5 Above IP

5.1 DNS and HTTP

The DNS (Domain Name System) [3] plays a crucial role in associating user-friendly string names with IP addresses. Leveraging our designed virtual interfaces, this process is automated seamlessly. Simultaneously, HTTP (Hypertext Transfer Protocol) [9] provides a standardized method for disseminating text and multimedia information across the Internet. While our virtual interface facilitates this task flawlessly, we have integrated a partially functioning TCP state machine for control handling [2], albeit less efficiently compared to the built-in TCP stack in the operating system.

5.2 Browsing the Web

With all the things we have accomplished, application level communication traffic can be easily handled by the Rathernet. Unfortunately, the bandwidth of the Rathernet is severely constrained, and we have to configure the route table to only allow the site we are visiting and the DNS server we are using to be directed to our virtual interface. Nevertheless, we eventually achieved visiting example.com with and without https encryption.

6 Difficulty and Challenge

6.1 Unreliable Hardware

The current sound card in use may not meet the perceived standards of quality. Transferring the USB connection between different computers can significantly impact its performance. Moreover, variations in volume levels can also influence the overall audio output, rendering selected parameters ineffective.

Our experiences suggest that a substantial amount of time is dedicated to grappling with subpar hardware. Numerous errors are attributable to hardware issues, and the sound card, in particular, is proving to be less dependable than desired. It is my hope that the course can either supply superior sound cards or relax the financial constraints on hardware purchases, enabling us to acquire more reliable options. The investment of time spent troubleshooting these issues is far more valuable than the savings achieved through budget constraints.

6.2 Hardware Inconsistency

We also observe that hardware configuration is critically important, with these constraints being intricately linked to the process.

1. The volume of every node must be set to a reasonable value (e.g., 40% in the Windows Control Panel), including the noise node.

2. The topology of the wires, connected as an equivalent graph, won't yield the same outcome if it deviates from being identical. We believe this is due to the system's resistance impacting the response.

Availability

To maintain reproducibility and help future development, we released our source code and data: https://github.com/mousany/rathernet.

References

[1] Wintun – layer 3 tun driver for windows.

[2] Internet Protocol. RFC 791, Sept. 1981.

[3] An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826, Nov. 1982.

[4] Domain names: Concepts and facilities. RFC 882, Nov. 1983.

[5] Carrier-sense medium access reference design, 2007.

[6] Rust programming language, 2021.

[7] Ieee standard for ethernet. *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)* (2022), 1–7025.

[8] BRADEN, R. T. Requirements for Internet Hosts - Communication Layers. RFC 1122, Oct. 1989.

[9] HIROSAKI, H., IMAI, H., AND OHTA, M. Multicarrier modulation for data transmission: an idea whose time has come. *IEEE Communications Magazine 33*, 5 (1995), 14–21.

[10] NIELSEN, H., FIELDING, R. T., AND BERNERS-LEE, T. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, May 1996.

[11] REED, I. S., AND SOLOMON, G. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics 8*, 2 (1960), 300–304.

[12] SUNSHINE. Understanding crc. *Sunshine's Homepage* (2006).

[13] VITERBI, A. J. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory 13*, 2 (April 1967), 260–269.

Notes

[1] The Viterbi algorithm implementaion is translated from a medium blog: https://gist.github.com/YairMZ/b88e594047c7b5366053cd7fb375a94f.

[2] The TCP state machine implementation is based on a youtube video: https://www.youtube.com/watch?v=$_{O}nJ9z-e_{T}Yt=0s$.