Rust for Network Servers

Synchronous and asynchronous network communication

Simon Pannek

Technical University of Munich
Munich, Germany

June 21, 2021

Introduction

Back in the 1960s, the protocols for one of the first computer networks were developed.

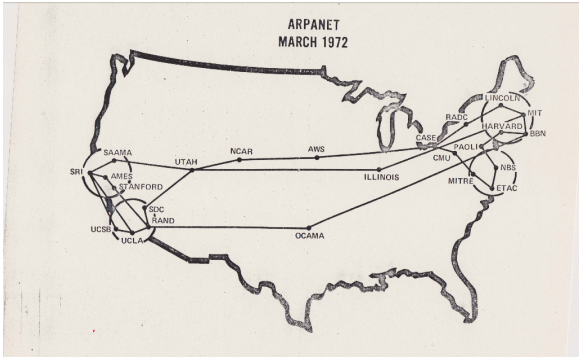

Figure: Ferris the Crab [1]

Introduction

What happened until now?

1. Additional security (encryption)

Introduction

What happened until now?

1. Additional security (encryption)
2. Introduction of new protocols

Introduction

What happened until now?

1. Additional security (encryption)
2. Introduction of new protocols
3. A lot was standardized (IEEE standards, W3C, ...)

What programming language to choose?

Introduction

```
void unsecure_function(void) {
    char buf[512];
    read_from_network(buf);

    ...
}
```

Buffer Overflow Attacks

```
void unsecure_function(void) {
    char buf[512];
    read_from_network(buf);

    ...
}
```

Introduction

```
def more_secure_function(socket):
    buf = socket.recv(512)

    ...
```

Interpreted languages

```
def more_secure_function(socket):
    buf = socket.recv(512)

    ...
```

Rust as a solution?

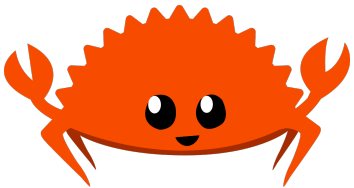

Figure: Ferris the Crab [3]

Ownership, borrowing, and lifetimes

Ownership, borrowing, and lifetimes

$\rightarrow$ No dangling references, illegal memory access, and memory leaks

Ownership, borrowing, and lifetimes

→ No dangling references, illegal memory access, and memory leaks

(Except if you are explicitly working with `unsafe` code)

The Transmission Control Protocol (TCP)

Networking in Rust

The Transmission Control Protocol (TCP)

- Active connection between two systems

Networking in Rust

The Transmission Control Protocol (TCP)

- Active connection between two systems
- Can be used to send a continuous stream of octets to another host

The Transmission Control Protocol (TCP)

- Active connection between two systems
- Can be used to send a continuous stream of octets to another host
- A sequence number is assigned to each octet in order to confirm it was received

The Transmission Control Protocol (TCP)

- Active connection between two systems
- Can be used to send a continuous stream of octets to another host
- A sequence number is assigned to each octet in order to confirm it was received
- If no acknowledgment (ACK) was received, the data is sent again

Networking in Rust

TCP client handling

```
let mut reader = BufReader::new(&stream);

loop {
    let mut buf = String::new();

    match reader.read_line(&mut buf) {
        ...
    }
}
```

Networking in Rust

TCP client handling

```
match reader.read_line(&mut buf) {
    Ok(0) => break,
    Ok(_) => print!("{}", buf),
    Err(e) => {
        eprintln!("{}", e);
        stream.shutdown(Both)?;
        break;
    }
}
```

Networking in Rust

TCP listener

```
let listener = TcpListener::bind(ADDRESS)
    .expect("Bind to address");

for stream in listener.incoming().flatten() {
    handle_client(stream)
        .expect("Handle client");
}
```

Networking in Rust

TCP client

```
let mut stream = TcpStream::connect(ADDRESS)?;

loop {
    let mut buf = String::new();

    match stdin().read_line(&mut buf) {
        ...
    }
}
```

Networking in Rust

TCP client

```
match stdin().read_line(&mut buf) {
    Ok(_) => {
        ...
    }
    Err(e) => {
        eprintln!("{}", e);
        stream.shutdown(Both)?;
        break;
    }
}
```

Networking in Rust

TCP client

```
            Ok(_) => {
                let bytes = buf.as_bytes();

                stream
                    .write_all(bytes)
                    .expect("Writing");
            }
```

The User Datagram Protocol (UDP)

Networking in Rust

The User Datagram Protocol (UDP)

- Commonly used if you do not need reliable delivery of streams

Networking in Rust

The User Datagram Protocol (UDP)

- Commonly used if you do not need reliable delivery of streams
- Tries to send messages to other programs with a minimal amount of protocol mechanism

Networking in Rust

The User Datagram Protocol (UDP)

- Commonly used if you do not need reliable delivery of streams
- Tries to send messages to other programs with a minimal amount of protocol mechanism
- No protection against package duplication

Networking in Rust

The User Datagram Protocol (UDP)

- Commonly used if you do not need reliable delivery of streams
- Tries to send messages to other programs with a minimal amount of protocol mechanism
- No protection against package duplication
- No checks whether the data sent has arrived and if it did in what order

Networking in Rust

UDP receiver

```
let socket = UdpSocket::bind(ADDRESS)?;

loop {
    let mut buf = [0u8; 1500];

    match socket.recv_from(&mut buf) {
        ...
    }
}
```

Networking in Rust

UDP receiver

```
match socket.recv_from(&mut buf) {
    Ok(_) => {
        let msg = from_utf8(&buf)
            .expect("Convert data");

        print!("{}", msg);
    }
    Err(e) => {
        eprintln!("{}", e);
        break;
    }
}
```

Networking in Rust

UDP sender

```
let socket = UdpSocket::bind("0.0.0.0:0")?;

loop {
    let mut buf = String::new();

    match stdin().read_line(&mut buf) {
        ...
    }
}
```

Networking in Rust

UDP sender

```
match stdin().read_line(&mut buf) {
    Ok(_) => {
        let bytes = buf.as_bytes();

        socket
            .send_to(bytes, ADDRESS)
            .expect("Sending");
    }
    Err(e) => {
        eprintln!("{}", e);
        break;
    }
}
```

Asynchronous Networking in Rust

Sequential programming	**Asynchronous programming**
- Blocking functions - Result is returned immediately - A new thread is required for each task that should run independently	- Non-blocking functions - Result is wrapped into futures - A scheduler dynamically assigns tasks to a limited amount of threads

Table: Comparison between sequential and asynchronous programming

The Tokio crate

The Tokio crate

- Tokio is a runtime for writing reliable network applications [2]

The Tokio crate

- Tokio is a runtime for writing reliable network applications [2]
- Tokio consists out of three major parts:

The Tokio crate

- Tokio is a runtime for writing reliable network applications [2]
- Tokio consists out of three major parts:
  1. I/O event loop

The Tokio crate

- Tokio is a runtime for writing reliable network applications [2]
- Tokio consists out of three major parts:
  1. I/O event loop
  2. Timer

The Tokio crate

- Tokio is a runtime for writing reliable network applications [2]
- Tokio consists out of three major parts:
  1. I/O event loop
  2. Timer
  3. Scheduler

Future demonstration

```rust
async fn async_function() {
    println!("Started task1");
    sleep(Duration::from_secs(5))
        .await;
    println!("Finished task1");
}
```

Asynchronous Networking in Rust

Future demonstration

```rust
#[tokio::main]
async fn main() {
    let future1 = async_function();

    let future2 = async {
        println!("Started task2");
        sleep(Duration::from_secs(3))
            .await;
        println!("Finished task2");
    };

    join!(future1, future2);
}
```

Asynchronous Networking in Rust

Asynchronous TCP client handling

```
tokio::spawn(async move {
    let mut reader =*BufReader::new(&mut stream);

    loop {
        let mut buf = String::new();

        match reader.read_line(&mut buf).await {
            ...
        }
    }

    Ok::<(), Error>(())
});
```

Asynchronous Networking in Rust

Asynchronous TCP client handling

```
match reader.read_line(&mut buf).await {
    Ok(0) => break,
    Ok(size) => print!("{}", buf),
    Err(e) => {
        eprintln!("{}", e);
        stream
            .shutdown().await?;
        break;
    }
}
```

Asynchronous TCP client handling

Asynchronous TCP listener

```
let listener = TcpListener::bind(ADDRESS).await?;

loop {
    let (stream, _) = listener.accept().await?;

    handle_client(stream);
}
```

Asynchronous Networking in Rust

Asynchronous TCP client

```
let mut stream = TcpStream::connect(ADDRESS).await?;

let mut reader = BufReader::new(stdin());

loop {
    let mut buf = String::new();

    match reader.read_line(&mut buf).await {
        ...
    }
}
```

Asynchronous Networking in Rust

Asynchronous TCP client

```
match reader.read_line(&mut buf).await {
    Ok(_) => {
        ...
    }
    Err(e) => {
        eprintln!("{}", e);
        stream.shutdown().await?;
        break;
    }
}
```

Asynchronous Networking in Rust

Asynchronous TCP client

```
        Ok(_) => {
            let bytes = buf.as_bytes();

            stream
                .write_all(bytes)
                .await?;
        }
```

What about Asynchronous UDP communication?

Network Client Showcase

Application Layer communication

OSI layers
Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Application Layer communication

OSI layers
Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Application Layer communication

Data struct

```
#[derive(Serialize, Deserialize, Debug)]
struct Data {
    number: u32,
    boolean: bool,
}
```

Application Layer communication

Warp HTTP server

```
let filter = warp::path!("data")
    .and(warp::post())
    .and(warp::body::json())
    .map(|data: Data| format!("Received: {:?}", data));

warp::serve(filter).run(ADDRESS).await;
```

Application Layer communication

Warp HTTP server

```
let client = reqwest::Client::new();

let response = client
    .post(URL)
    .json(&Data {
        number: 5,
        boolean: true,
    })
    .send()
    .await
    .expect("Sending request");

println!("{}", response.text().await.expect("Get text"));
```

Application Layer communication

Client response

```
Received: Data { number: 5, boolean: true }
```

Conclusion

Rust is a very powerful language for writing network applications

Conclusion

Are there any questions?

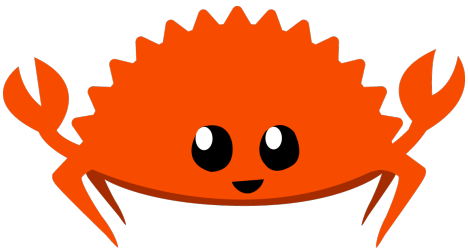

Figure: Ferris the Crab [3]

References

UCLA and BBN (1972) [1]

The map of ARPANET in March 1972.

https://commons.wikimedia.org/wiki/File:Arpanet_1972_Map.png

Docs.rs/tokio [2]

Tokio crate documentation.

https://docs.rs/tokio/1.6.1/tokio/

Rustacean.net [3]

Ferris the Crab.

https://rustacean.net/assets/rustacean-flat-happy.png