

**LAPORAN**  
**Sistem dan Jaringan Komputer**  
**Sistem Forwarding Pesan ke Email dan Web Server**  
**Berbasis Socket TCP**



**DISUSUN OLEH :**

**KELOMPOK 9**

Muqsith Barru Pamungkas	(2423600035)
Miftakhul Zannah	(2423600046)
Andika Rifan Rafiudins	(2423600055)

**DOSEN :**

Prof. Ir. Amang Sudarsono S.T., Ph.D

**PROGRAM STUDI TEKNOLOGI REKAYASA INTERNET**  
**DEPARTEMEN TEKNIK ELEKTRO**  
**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA**  
**2024/2025**

## ABSTRAK

Proyek ini bertujuan untuk mengembangkan dan mengimplementasikan sistem message forwarding berbasis protokol TCP dan teknologi Node.js untuk mendukung pengiriman pesan melalui email serta penampilannya pada halaman web. Sistem ini dirancang dalam lingkungan jaringan lokal (LAN) dan terdiri dari tiga komponen utama: client, forwarding server, dan email server. Dalam alur kerjanya, client mengirimkan data berupa pesan dan alamat email yang diterima oleh forwarding server melalui protokol TCP. Forwarding server memproses pesan tersebut dan meneruskannya ke dua jalur utama: server email untuk dikirimkan ke alamat tujuan melalui protokol SMTP menggunakan format standar MIME, dan interface untuk menampilkan pesan secara dinamis di halaman web. Pengujian sistem menunjukkan bahwa komunikasi antara client dan forwarding server melalui socket TCP/UDP berjalan stabil dan efisien, dengan latensi yang rendah. Pesan yang dikirimkan berhasil diteruskan ke server email untuk dikirimkan ke alamat tujuan, dan dapat ditampilkan secara real-time di halaman web. Integrasi Interface memungkinkan pesan ditampilkan secara terorganisir dalam antarmuka web yang mudah diakses. Dengan keberhasilan ini, sistem dapat berfungsi sebagai prototipe dasar untuk kebutuhan komunikasi berbasis jaringan, baik di lingkungan akademik maupun industri, khususnya dalam skenario pengelolaan pesan di jaringan lokal.

Kata kunci: Message forwarding, TCP, Interface, Email server, Web-based messaging, komunikasi jaringan.

# BAB I

## PENDAHULUAN

### 1.1 Latar Belakang

Pemahaman tentang sistem komunikasi client-server menjadi salah satu kompetensi yang harus dikuasai dalam pembelajaran teknologi informasi. Salah satu implementasi penting dari sistem ini adalah bagaimana mengintegrasikan berbagai layanan seperti email server dan web server untuk mendukung pengiriman dan penampilan pesan secara terstruktur dan efisien. Untuk mencapai hal tersebut, diperlukan pemahaman mendalam mengenai protokol komunikasi seperti TCP/UDP yang digunakan untuk menjalin koneksi antara client dan server, serta mekanisme forwarder server yang berfungsi sebagai perantara dalam pengiriman pesan ke layanan tujuan.

Dalam konteks ini, pengiriman pesan ke email server melalui protokol email client-server menjadi salah satu metode utama untuk mendistribusikan informasi langsung ke penerima. Di sisi lain, penggunaan Interface memungkinkan web server menampilkan data secara dinamis, sehingga pesan yang diterima dari client dapat diakses dan dilihat. Tugas ini bertujuan untuk memberikan pemahaman mendalam tentang cara kerja komponen-komponen tersebut dan bagaimana mengintegrasikannya dalam sebuah sistem yang andal.

Melalui tugas ini, diharapkan mahasiswa mampu mengaplikasikan teori yang telah dipelajari ke dalam praktik, serta memiliki kemampuan analisis untuk merancang sistem yang efisien dan sesuai kebutuhan. Hal ini tidak hanya bermanfaat dalam konteks akademik, tetapi juga sebagai persiapan menghadapi tantangan dunia kerja di bidang teknologi informasi dan komunikasi.

### 1.2 Masalah

1. Bagaimana cara membangun sistem pengiriman pesan antara client dan server forwarder menggunakan komunikasi socket TCP?
2. Bagaimana cara memforward pesan dari client melalui forwarder server ke email server menggunakan pengiriman email melalui protokol SMTP?
3. Bagaimana cara memforward pesan dari client ke web server dan menampilkan pesan tersebut di halaman interface web?

### 1.3 Solusi dan Rencana Realisasi Solusi

Untuk membangun sistem pengiriman pesan antara client dan server forwarder menggunakan komunikasi socket TCP, langkah pertama adalah membuat server socket di sisi server yang dapat menerima koneksi dari client. Server ini akan mendengarkan pada port tertentu dan menerima pesan yang dikirimkan oleh client melalui koneksi TCP. Di sisi client, aplikasi yang dibangun akan menghubungkan ke server menggunakan alamat IP dan port yang sesuai, kemudian mengirimkan pesan melalui koneksi yang terjalin. Setelah pesan diterima oleh server forwarder, server akan memprosesnya dan meneruskan pesan tersebut ke tujuan yang sesuai, apakah itu email server atau web server. Selama proses ini, penting untuk mengelola koneksi dengan memastikan stabilitas dan penanganan error yang tepat, seperti pengaturan timeout yang optimal dan pengecekan status koneksi untuk memastikan pesan dikirim dengan lebih efisien.

Untuk memforward pesan dari client melalui forwarder server ke email server menggunakan pengiriman email melalui protokol SMTP, setelah server forwarder menerima pesan dari client, server harus terhubung dengan email server menggunakan SMTP. Server SMTP harus mengikuti standart dari rfc 5321. Server forwarder akan menggunakan pustaka SMTP seperti 'smtplib' di Python untuk membuat koneksi dengan server email, dan kemudian mengirimkan pesan yang diterima dari client sebagai email. Pada tahap ini, server forwarder akan mengirimkan pesan client dengan menggunakan informasi SMTP, termasuk alamat server, dan port. Selama proses pengiriman, penting untuk menangani kemungkinan error, seperti kegagalan koneksi atau kredensial

yang salah, untuk memastikan bahwa email berhasil dikirim dan diterima dengan benar oleh email server tujuan.

Selanjutnya, untuk memforward pesan dari client ke web server berbasis Node.js dan menampilkan pesan tersebut di halaman web, server forwarder akan mengonversi pesan yang diterima menjadi format yang dapat diproses oleh web server, seperti JSON. Setelah itu, server forwarder mengirimkan permintaan HTTP ke web server dengan membawa pesan dalam body permintaan. Di sisi web server, aplikasi berbasis Node.js akan menangani permintaan yang masuk, membaca data yang dikirimkan, dan menyimpan atau menampilkan pesan tersebut di halaman web. Node.js akan menyajikan pesan dalam format HTML yang dapat diakses oleh browser pengguna. Selama proses ini, penting untuk memastikan keamanan dengan melakukan validasi dan sanitasi pada pesan yang diterima untuk mencegah potensi serangan seperti XSS (Cross-Site Scripting) atau injection.

## BAB II

# PERANCANGAN DAN KONFIGURASI JARINGAN

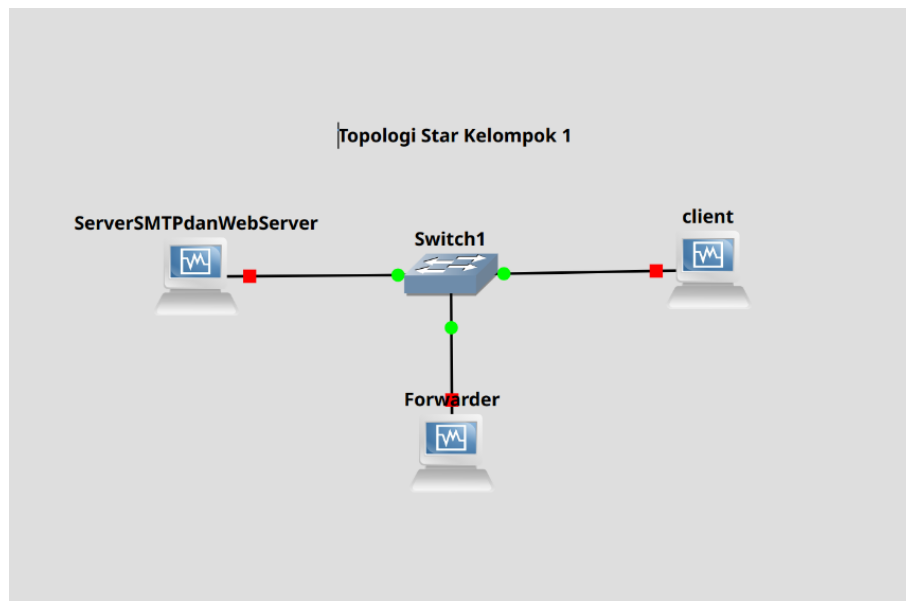


Figure 1: Topologi Star Sederhana

Dalam proyek ini kami merancang dengan menggunakan topologi star sederhana. Di dalam topologi ini, kami menggunakan tiga virtual machine yaitu untuk client, forwarder, dan server. Dari masing masing virtual machine tersebut didalamnya berisi os linux yang saling terhubung satu sama lain dengan menggunakan jaringan lokal dengan ip class c. Untuk cara kerjanya sendiri yaitu, client digunakan untuk mengirimkan email ke forwarder menggunakan socket TCP, kemudian dari forwarder akan ditangkap dan diteruskan ke server dengan protokol SMTP.

### 2.1 Rincian Jaringan

No.	Nama PC	IP Address	Subnet Mask	Operation System	Protokol
1	Client	192.168.50.1	255.255.255.0	Linux OS (Kali Linux)	TCP
2	Forwarder	192.168.50.2	255.255.255.0	Linux OS (Debian 12)	TCP,SMTP
3	Server	192.168.50.3	255.255.255.0	Linux OS (Debian 12)	SMTP

Table 1: Tabel Rincian Jaringan

## 2.2 Konfigurasi PC Client

### 1. Konfigurasi Jaringan

Untuk konfigurasi jaringan, digunakan metode 'inet static' pada file '/etc/network/interfaces'. Berikut adalah contoh konfigurasi yang digunakan: Konfigurasi tersebut mengatur alamat IP

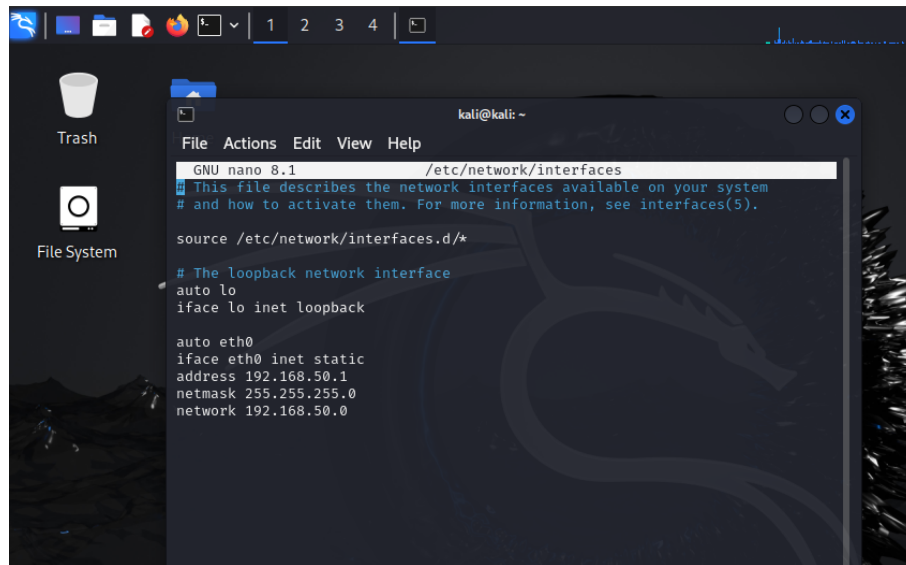


Figure 2: Konfigurasi IP Address

secara statis agar PC client terhubung dengan jaringan lokal menggunakan IP yang telah ditentukan yaitu 192.168.50.1

### 2. Cara Mengirimkan Email

Pada PC Client, untuk mengirimkan email ke forwarder, kami menggunakan socket TCP yang dikembangkan dengan bahasa pemrograman Rust. Untuk menjalankannya, kami menggunakan command cargo run, seperti yang dapat dilihat pada gambar di bawah ini.

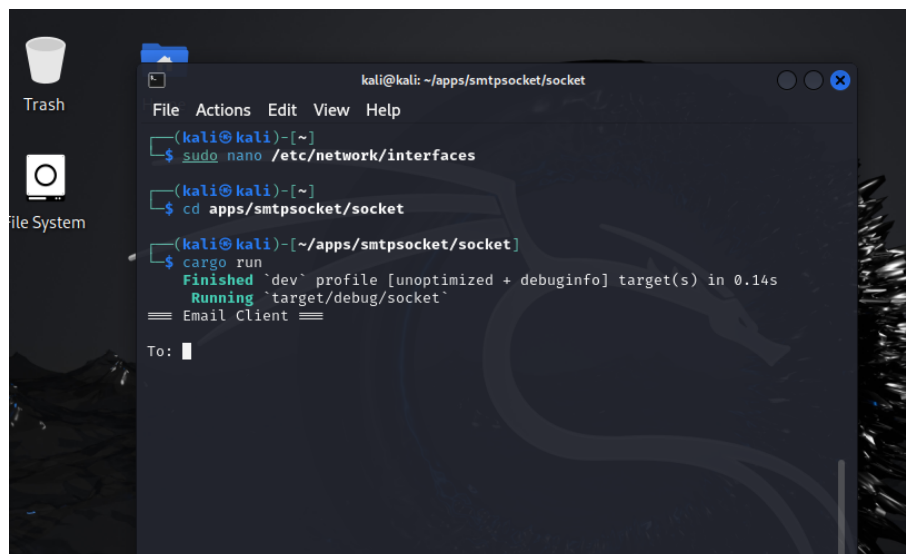


Figure 3: run program pc client

### 3. Code Client

```
1 use serde::Serialize;
2 use std::io::{self, Read, Write};
3 use std::net::TcpStream;
```

4

```

5  #[derive(Serialize)]
6  struct Email {
7      sender: String,
8      recipient: String,
9      subject: String,
10     body: String,
11 }
12
13 fn send_message(email: &str) {
14     let server = "192.168.50.2:1239";
15
16     match TcpStream::connect(server) {
17         Ok(mut stream) => {
18             println!("Successfully connected to forwarder {}", server);
19             match stream.write_all(email.as_bytes()) {
20                 Ok(_) => {
21                     println!("Try to sent email to forwarder");
22                 }
23                 Err(e) => {
24                     println!("Failed to send email to forwarder {}", e);
25                 }
26             }
27             let mut response = String::new();
28             match stream.read_to_string(&mut response) {
29                 Ok(_) => println!("Server response: {}", response),
30                 Err(e) => eprintln!("Failed to read server response: {}", e),
31             }
32         }
33         Err(e) => {
34             println!("Failed to connect to forwarder {}", e)
35         }
36     }
37 }
38
39 fn main() {
40     let sender = "ketua@kelompoksatu.com";
41     let mut recipient = String::new();
42     let mut subject = String::new();
43     let mut body = String::new();
44
45     println!("=== Email Client ===\n");
46
47     loop {
48         print!("To: ");
49         io::stdout().flush().unwrap();
50         io::stdin().read_line(&mut recipient).unwrap();
51         recipient = recipient.trim().to_string();
52         print!("Subject: ");
53         io::stdout().flush().unwrap();
54         io::stdin().read_line(&mut subject).unwrap();
55         subject = subject.trim().to_string();
56         print!("body: ");
57         io::stdout().flush().unwrap();
58         io::stdin().read_line(&mut body).unwrap();

```

```

59     body = body.trim().to_string();
60
61     let email = Email {
62         sender: sender.to_string(),
63         recipient: recipient.clone(),
64         subject: subject.clone(),
65         body: body.clone(),
66     };
67
68     let email_json = serde_json::to_string(&email).unwrap();
69     // println!("email: {}", email_json);
70     send_message(&email_json);
71     let mut choose = String::new();
72     print!("LANJUT KIRIM LAGI ? y/n: ");
73     io::stdout().flush().unwrap();
74     io::stdin().read_line(&mut choose).unwrap();
75     choose = choose.trim().to_string();
76     if choose.eq_ignore_ascii_case("n") {
77         break;
78     }
79     recipient.clear();
80     body.clear();
81 }
82 println!("Seeee yoooooooooooooooooooo");
83 }

```

## 2.3 Konfigurasi PC forwarder

### 1. Konfigurasi Jaringan

Pada pc forwarder juga dilakukan pengkonfigurasian ip static pada file ‘/etc/network/interfaces’, untuk lebih jelasnya bisa dilihat pada gambar dibawah ini. Konfigurasi tersebut mengatur ala-

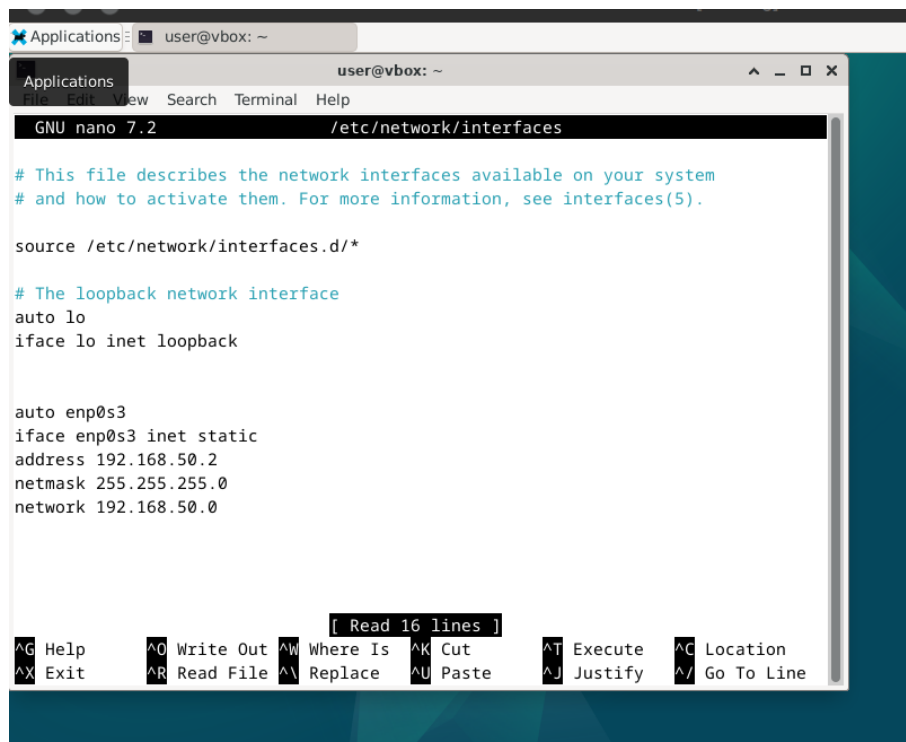


Figure 4: Konfigurasi IP Address



mat IP secara statis agar PC forwarder ini dapat terhubung dengan jaringan lokal menggunakan IP yang telah ditentukan yaitu 192.168.50.2

## 2. Cara Menjalakan Forwarder

Pada pc forwarder kami menangkap data dari client yang akan diteruskan ke pc server. Untuk programnya sendiri kami mengembangkannya dengan bahasa python sehingga untuk menjalankannya seperti gambar dibawah ini.

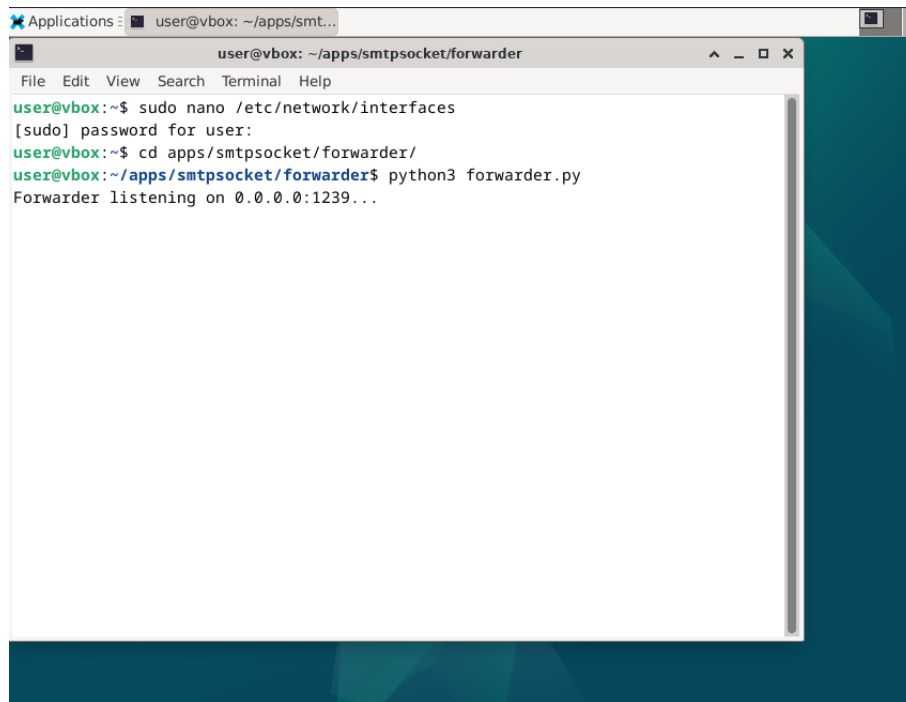


Figure 5: run program pc forwarder

## 3. Code Forwarder

```
1  import socket
2  import json
3  from threading import Thread
4  import smtplib
5  from email.mime.text import MIMEText
6  from email.mime.multipart import MIMEMultipart
7
8  def send_email_via_smtp(email_data):
9      smtp_server = "192.168.50.3" # IP dari PC3 (SMTP Server)
10     smtp_port = 25 # Port yang sama seperti yang digunakan di
11                    ↪ server SMTP PC3
12
13     try:
14         # Membuat pesan email
15         msg = MIMEMultipart()
16         msg['From'] = email_data['sender']
17         msg['To'] = email_data['recipient']
18         msg['Subject'] = email_data['subject']
19         msg.attach(MIMEText(email_data['body'], 'plain'))
20
21         # Mengirim email melalui SMTP (PC3)
22         with smtplib.SMTP(smtp_server, smtp_port) as server:
23             server.sendmail(email_data['sender'], email_data['recipient'],
24                             ↪ msg.as_string())
```

```

23
24     # print("Email successfully sent via SMTP.")
25     return "Email successfully sent via SMTP."
26 except Exception as e:
27     # print(f"Error sending email via SMTP: {e}")
28     error_message = f"Error sending email via SMTP error message: {e}"
29     return error_message
30
31 def handle_client(client_socket):
32     try:
33         data = client_socket.recv(1024).decode()
34         email_data = json.loads(data.replace("'", "\'")) # Parse the string as
35         ↪ JSON
36         print(f"Received email data: {email_data}")
37         result = send_email_via_smtp(email_data) # Kirim email via SMTP (ke
38         ↪ PC3)
39         print(f"iki lo hasil e {result.encode()}")
40         client_socket.sendall(result.encode())
41
42     except Exception as e:
43         print(f"Error handling client: {e} {result}")
44         client_socket.sendall(b"Failed to forward email".encode())
45     finally:
46         client_socket.close()
47
48 def start_forwarder(server_ip, server_port):
49     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
50         server_socket.bind((server_ip, server_port))
51         server_socket.listen(5)
52         print(f"Forwarder listening on {server_ip}:{server_port}...")
53         while True:
54             client_socket, addr = server_socket.accept()
55             print(f"Connection established with {addr}")
56             thread = Thread(target=handle_client, args=(client_socket,))
57             thread.start()
58
59 if __name__ == "__main__":
60     SERVER_IP = "0.0.0.0"
61     SERVER_PORT = 1239
62     start_forwarder(SERVER_IP, SERVER_PORT)

```

## 2.4 Konfigurasi PC Server

### 1. Konfigurasi Jaringan

Pada pc server juga dilakukan pengkonfigurasian ip static pada file ‘/etc/network/interfaces’, untuk lebih jelasnya bisa dilihat pada gambar dibawah ini. Konfigurasi tersebut mengatur alamat IP secara statis agar PC server ini dapat terhubung dengan jaringan lokal menggunakan IP yang telah ditentukan yaitu 192.168.50.3

### 2. Cara Menjalakan Server

Pada pc server kami menjalan dua program yaitu program SMTP server yang kami bangun dari bahasa rust dan web server yang kami bangun dari nodejs. Untuk perintah menjalankannya sendiri dapat dilihat pada gambar dibawah ini.

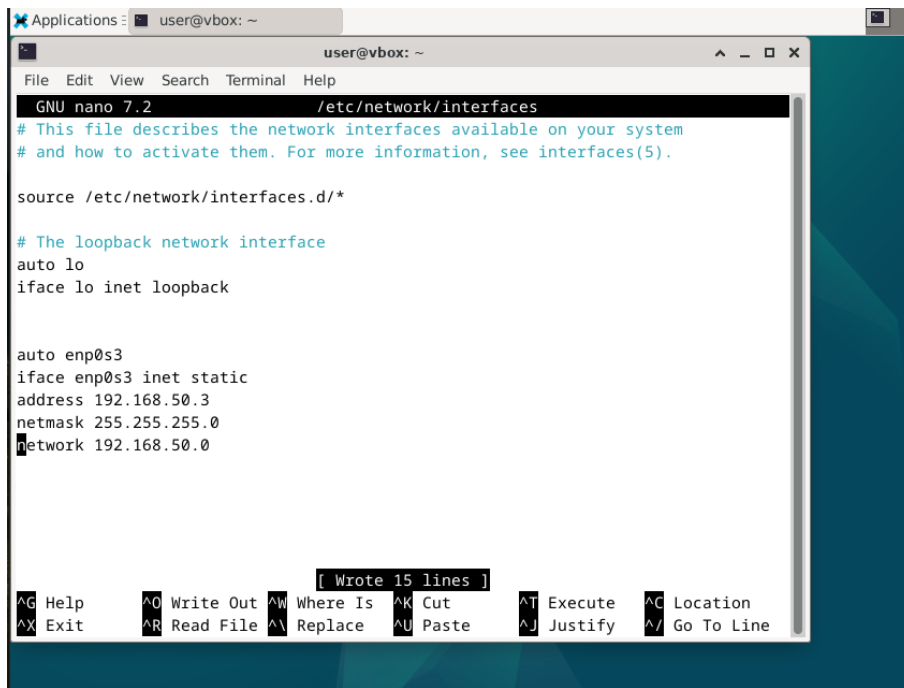


Figure 6: Konfigurasi IP Address

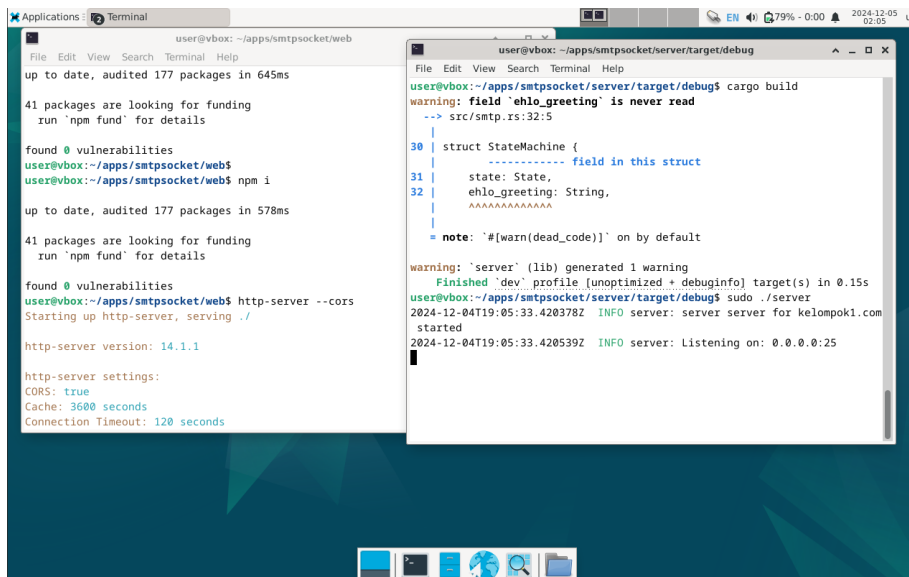


Figure 7: run program pc server

### 3. Code Server main.rs

```

1 use anyhow::{Context, Result};
2 use tokio::net::TcpListener;
3
4 use std::env;
5
6 use server::smtp;
7
8 #[tokio::main]
9 async fn main() -> Result<()> {
10     tracing_subscriber::fmt::init();
11
12     let addr = env::args()
13         .nth(1)
14         .unwrap_or_else(|| "0.0.0.0:25".to_string());
  
```

```

15
16     let domain = &env::args()
17         .nth(2)
18         .unwrap_or_else(|| "kelompok1.com".to_string());
19
20     tracing::info!("server server for {domain} started");
21
22     let listener = TcpListener::bind(&addr).await?;
23     tracing::info!("Listening on: {}", addr);
24
25     // Task for deleting old mail
26     // periodically_clean_db(tokio::time::Duration::from_secs(3600));
27
28     // Main loop: accept connections and spawn a task to handle them
29     loop {
30         let (stream, addr) = listener.accept().await?;
31         tracing::info!("Accepted a connection from {}", addr);
32
33         tokio::task::LocalSet::new()
34             .run_until(async move {
35                 let smtp = smtp::Server::new(domain, stream).await?;
36                 tokio::time::timeout(std::time::Duration::from_secs(300),
37                     ↪ smtp.serve())
38                     .await
39                     .context("connection timed out")
40             })
41             .await
42             .ok();
43     }

```

#### 4. Code Server smtp.rs

```

1     use anyhow::{Context, Result};
2     use regex::Regex;
3     use serde::{Deserialize, Serialize};
4     use serde_json::{json, Value};
5     use std::fs::{self, OpenOptions};
6     use tokio::io::{AsyncReadExt, AsyncWriteExt};
7
8     #[derive(Clone, Debug, Default, PartialEq, Eq, Serialize, Deserialize)]
9     pub struct Mail {
10         pub from: String,
11         pub to: Vec<String>,
12         pub data: String,
13     }
14
15     #[derive(Clone, Debug, PartialEq, Eq)]
16
17     enum State {
18         Fresh,
19         Greeted,
20         ReceivingRcpt(Mail),
21         ReceivingData(Mail),
22         Received(Mail),

```

```

23 }
24
25 struct StateMachine {
26     state: State,
27     ehlo_greeting: String,
28 }
29
30 impl StateMachine {
31     const OH_HAI: &'static [u8] = b"220 SMTP SERVER KELOMPOK 1 TRI B \n";
32     const KTHXBYE: &'static [u8] = b"221 Bye\n";
33     const HOLD_YOUR_HORSES: &'static [u8] = &[];
34
35     pub fn new(domain: impl AsRef<str>) -> Self {
36         let domain = domain.as_ref();
37         let ehlo_greeting = format!("250-{{domain}} Hello {{domain}}\n250 AUTH
38             ↪ PLAIN LOGIN\n");
39         Self {
40             state: State::Fresh,
41             ehlo_greeting,
42         }
43     }
44
45     pub fn handle_smtp(&mut self, raw_msg: &str) -> Result<&[u8]> {
46         // let re_email =
47         ↪ Regex::new(r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$").unwrap();
48         let re_email =
49         ↪ Regex::new(r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}>$").unwrap();
50
51         const MAX_MSG_SIZE: usize = 10 * 1024 * 1024; // 10 MB
52
53         tracing::trace!("Received {{raw_msg}} in state {:?}", self.state);
54         let mut msg = raw_msg.split_whitespace();
55         let command = msg.next().context("received empty
56             ↪ command")?.to_lowercase();
57         let state = std::mem::replace(&mut self.state, State::Fresh);
58
59         match (command.as_str(), state) {
60             ("ehlo", State::Fresh) => {
61                 tracing::trace!("Sending AUTH info and SIZE support");
62                 self.state = State::Greeted;
63                 Ok(b"250-Hello\r\n250-SIZE 10485760\r\n250 AUTH\r\n") // 10MB
64                 ↪ size limit
65             }
66             ("helo", State::Fresh) => {
67                 tracing::trace!("Received HELO");
68                 self.state = State::Greeted;
69                 Ok(b"250 Hello\r\n")
70             }
71             ("noop", _) | ("help", _) => {
72                 tracing::trace!("Got {{command}}");
73                 Ok(b"250 OK\r\n")
74             }
75             ("rset", _) => {
76                 tracing::trace!("Resetting state");

```

```

72         self.state = State::Fresh;
73         Ok(b"250 OK\r\n")
74     }
75     ("mail", State::Greeted) => {
76         tracing::trace!("Receiving MAIL");
77         let from = msg.next().context("received empty MAIL")?;
78         let from = from
79             .strip_prefix("FROM:")
80             .context("received incorrect MAIL")?
81             .trim()
82             .to_string();
83         if re_email.is_match(&from) {
84             tracing::debug!("Valid MAIL FROM: {from}");
85             self.state = State::ReceivingRcpt(Mail {
86                 from: from.to_string(),
87                 ..Default::default()
88             });
89             Ok(b"250 OK\r\n")
90         } else {
91             tracing::warn!("Invalid MAIL FROM address: {from}");
92             Ok(b"501 Syntax: Invalid email address\r\n")
93         }
94     }
95     ("rcpt", State::ReceivingRcpt(mut mail)) => {
96         tracing::trace!("Receiving RCPT");
97         let to = msg.next().context("received empty RCPT")?;
98         let to = to
99             .strip_prefix("TO:")
100             .context("received incorrect RCPT")?
101             .trim();
102         if re_email.is_match(to) {
103             tracing::debug!("Valid RCPT TO: {to}");
104             mail.to.push(to.to_lowercase());
105             self.state = State::ReceivingRcpt(mail);
106             Ok(b"250 OK\r\n")
107         } else {
108             tracing::warn!("Invalid RCPT TO address: {to}");
109             Ok(b"501 Syntax: Invalid recipient address\r\n")
110         }
111     }
112     ("data", State::ReceivingRcpt(mail)) => {
113         if mail.to.is_empty() {
114             tracing::warn!("DATA command received without RCPT");
115             Ok(b"503 Error: RCPT TO command must precede DATA\r\n")
116         } else {
117             tracing::trace!("Ready to receive DATA");
118             self.state = State::ReceivingData(mail);
119             Ok(b"354 Start mail input; end with <CR><LF>.<CR><LF>\r\n")
120         }
121     }
122     ("quit", State::ReceivingData(mail)) => {
123         self.state = State::Received(mail);
124         Ok(StateMachine::KTHXBYE)
125     }

```

```

126         (_, State::ReceivingData(mut mail)) => {
127             tracing::trace!("Appending data");
128             if raw_msg.ends_with("\r\n.\r\n") {
129                 let trimmed_data = raw_msg.trim_end_matches("\r\n.\r\n");
130                 mail.data.push_str(trimmed_data);
131                 if mail.data.len() > MAX_MSG_SIZE {
132                     tracing::warn!("Message size exceeds limit");
133                     Ok(b"552 Error: Message size exceeds maximum size\r\n")
134                 } else {
135                     tracing::trace!(
136                         "Email received: FROM: {} TO: {:?} DATA: {}",
137                         mail.from,
138                         mail.to,
139                         mail.data
140                     );
141                     mail.data += raw_msg;
142                     self.state = State::ReceivingData(mail);
143                     Ok(b"250 OK\r\n")
144                 }
145             } else {
146                 mail.data.push_str(raw_msg);
147                 self.state = State::ReceivingData(mail);
148                 Ok(b"250 Continue\r\n")
149             }
150         }
151     _ => {
152         tracing::warn!("Unexpected message: {raw_msg}");
153         Ok(b"500 Syntax error, command unrecognized\r\n")
154     }
155 }
156 }
157
158 }
159
160 pub struct Server {
161     stream: tokio::net::TcpStream,
162     state_machine: StateMachine,
163 }
164
165 impl Server {
166     pub async fn new(domain: impl AsRef<str>, stream: tokio::net::TcpStream) ->
167         Result<Self> {
168         Ok(Self {
169             stream,
170             state_machine: StateMachine::new(domain),
171         })
172     }
173
174     pub async fn serve(mut self) -> Result<()> {
175         self.greet().await?;
176
177         let mut buf = vec![0; 1024 * 1024];
178         loop {
179             let n = self.stream.read(&mut buf).await?;

```

```

179
180         if n == 0 {
181             self.state_machine.handle_smtp("quit").ok();
182             break;
183         }
184         let msg = std::str::from_utf8(&buf[0..n])?;
185         let response = self.state_machine.handle_smtp(msg)?;
186         if response != StateMachine::HOLD_YOUR_HORSES {
187             self.stream.write_all(response).await?;
188         }
189         if response == StateMachine::KTHXBYE {
190             break;
191         }
192     }
193
194     match self.state_machine.state {
195         State::Received(ref mail) => {
196             self.save_email_to_json(mail).await?;
197         }
198         State::ReceivingData(ref mail) => {
199             self.save_email_to_json(mail).await?;
200         }
201         _ => {}
202     }
203     Ok(())
204 }
205
206 async fn save_email_to_json(&self, mail: &Mail) -> Result<()> {
207     let file_path = "received_email.json";
208     let email_entry = json!({
209         "sender": mail.from,
210         "recipient": mail.to,
211         "subject": self.extract_subject(&mail.data)?,
212         "body": self.extract_body(&mail.data)?,
213     });
214
215     let mut emails: Vec<Value> = if let Ok(existing_content) =
216         ↪ fs::read_to_string(file_path) {
217         serde_json::from_str(&existing_content)?
218     } else {
219         Vec::new()
220     };
221
222     emails.push(email_entry);
223
224     let file = OpenOptions::new()
225         .write(true)
226         .create(true)
227         .truncate(true)
228         .open(file_path)?;
229     serde_json::to_writer_pretty(file, &emails)?;
230     println!("Email saved to {}", file_path);
231
232     Ok(())

```



```

232     }
233
234     fn extract_subject(&self, data: &str) -> Result<String> {
235         for line in data.lines() {
236             if line.to_lowercase().starts_with("subject:") {
237                 return Ok(line[8..].trim().to_string());
238             }
239         }
240         Ok("No Subject".to_string())
241     }
242
243     fn extract_body(&self, data: &str) -> Result<String> {
244         if let Some(boundary_start) = data.find("-----") {
245             let body_start = data[boundary_start..]
246                 .find("\r\n\r\n")
247                 .map(|pos| boundary_start + pos + 4)
248                 .unwrap_or(data.len());
249             let body_end = data[body_start..]
250                 .find("\r\n-----")
251                 .map(|pos| body_start + pos)
252                 .unwrap_or(data.len());
253             return Ok(data[body_start..body_end].trim().to_string());
254         }
255
256         if let Some(headers_end) = data.find("\r\n\r\n") {
257             return Ok(data[headers_end + 4..].trim().to_string());
258         }
259
260         Ok("No Body Found".to_string())
261     }
262
263     async fn greet(&mut self) -> Result<()> {
264         self.stream
265             .write_all(StateMachine::OH_HAI)
266             .await
267             .map_err(|e| e.into())
268     }
269 }
270
271 #[cfg(test)]
272 mod tests {
273     use super::*;
274
275     #[test]
276     fn test_regular_flow() {
277         let mut sm = StateMachine::new("dummy");
278         sm.handle_smtp("HELO localhost").unwrap();
279         sm.handle_smtp("MAIL FROM: <local@example.com>").unwrap();
280         sm.handle_smtp("RCPT TO: <a@localhost.com>").unwrap();
281         sm.handle_smtp("RCPT TO: <b@localhost.com>").unwrap();
282         sm.handle_smtp("DATA hello world\n").unwrap();
283         sm.handle_smtp("QUIT").unwrap();
284     }
285

```

```

286     #[test]
287     fn test_no_greeting() {
288         let mut sm = StateMachine::new("dummy");
289         for command in [
290             "MAIL FROM: <local@example.com>",
291             "RCPT TO: <local@example.com>",
292             "DATA hey",
293             "GARBAGE",
294         ] {
295             assert!(sm.handle_smtp(command).is_err());
296         }
297     }
298 }
299
300 #[test]
301 fn test_no_greeting() {
302     let re_email =
303         ↪ Regex::new(r"^<([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,})>$").unwrap()
304     let email = "<ketua@kelompoksatu.com>";
305     println!("Valid email: {}", re_email.is_match(email));
306 }

```

# BAB III

## SIMULASI DAN HASIL TEST-BED

### 3.1 Pengujian Antar PC

1. Pada saat client mengirim, forwarder dan server mati

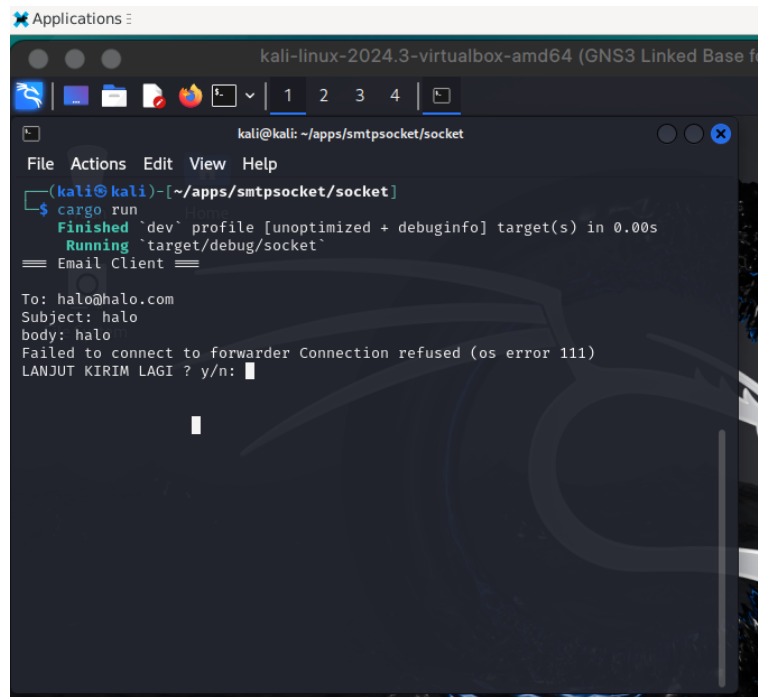


Figure 8: Client hidup, forwarder dan server mati

2. Pada saat PC client dan forwarder hidup, pc Server mati

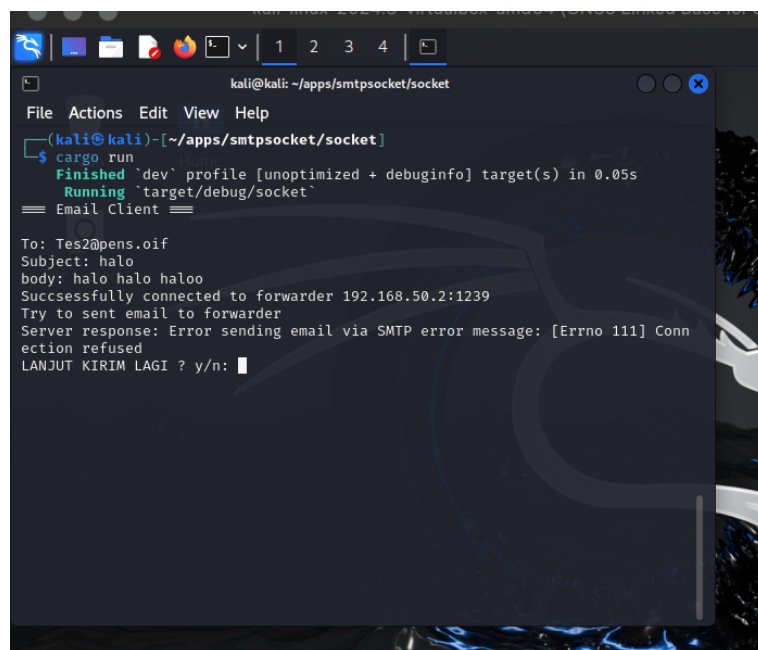
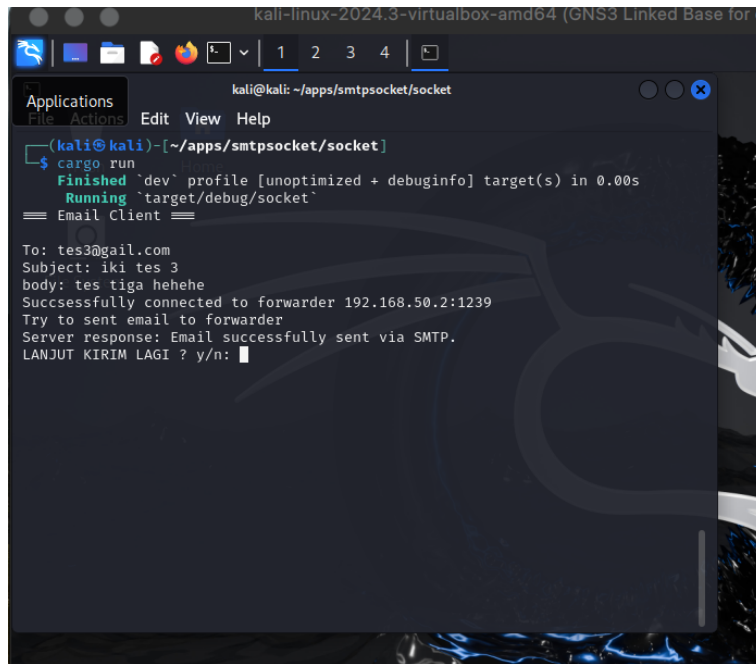


Figure 9: Client forwarder hidup dan server mati

3. Pada saat semua pc hidup

A terminal window titled 'kali@kali: ~/apps/smtpsocket/socket' showing the output of a 'cargo run' command. The output indicates that an email was successfully sent via SMTP to 'tes3@gmail.com' with the subject 'iki tes 3' and body 'tes tiga hehehe'. The terminal also shows the email client interface with fields for 'To', 'Subject', and 'body', and a status message 'Successfully connected to forwarder 192.168.50.2:1239'.

```
kali@kali: ~/apps/smtpsocket/socket
(kali@kali)-[~/apps/smtpsocket/socket]
$ cargo run
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
Running `target/debug/socket`

Email Client

To: tes3@gmail.com
Subject: iki tes 3
body: tes tiga hehehe
Successfully connected to forwarder 192.168.50.2:1239
Try to sent email to forwarder
Server response: Email successfully sent via SMTP.
LANJUT KIRIM LAGI ? y/n: 
```

Figure 10: semua pc hidup

### 3.2 Hasil pada webserver ketika berhasil

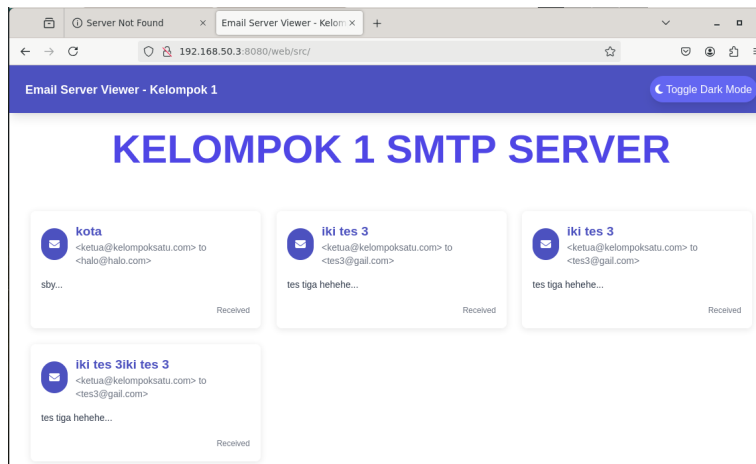


Figure 11: Hasil tampilan web server

### 3.3 Test Menggunakan telnet dari pc client ke SMTP server

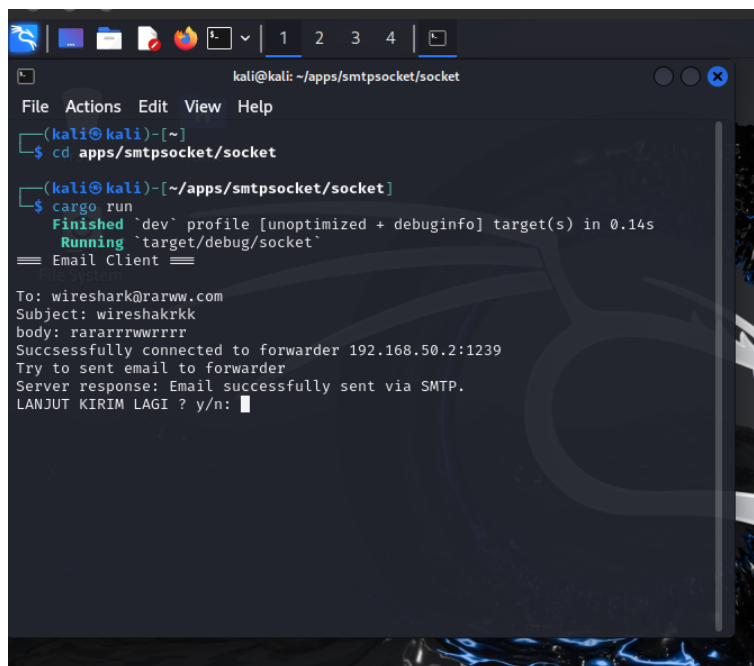
```
500 Syntax error, command unrecognized
Connection closed by foreign host.

(kali@kali)-[~/apps/smtpsocket/socket]
$ telnet 192.168.50.3 25
Trying 192.168.50.3...
Connected to 192.168.50.3.
Escape character is '^]'.
220 SMTP SERVER KELOMPOK 1 TRI B
EHLO localhost
500 Syntax error, command unrecognized
EHLO localhost
250-Hello
250-SIZE 10485760
250 AUTH
helo
500 Syntax error, command unrecognized
HELO
250 Hello
mail FROM:<tri@tri.com>
250 OK
rcpt TO:<pens@pens.id>
250 OK
data
354 Start mail input; end with <CR><LF>.<CR><LF>
```

Figure 12: test dengan telnet

### 3.4 Hasil Capture dari Wireshark

#### 1. Data yang dikirim



```
kali@kali: ~/apps/smtpsocket/socket
File Actions Edit View Help
(kali@kali)-[~]
$ cd apps/smtpsocket/socket
(kali@kali)-[~/apps/smtpsocket/socket]
$ cargo run
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.14s
Running `target/debug/socket`
== Email Client ==
To: wireshark@rarww.com
Subject: wireshakrkk
body: rararrrwwrrrr
Successfully connected to forwarder 192.168.50.2:1239
Try to sent email to forwarder
Server response: Email successfully sent via SMTP.
LANJUT KIRIM LAGI ? y/n: █
```

Figure 13: data yang dikirim

## 2. Socket TCP

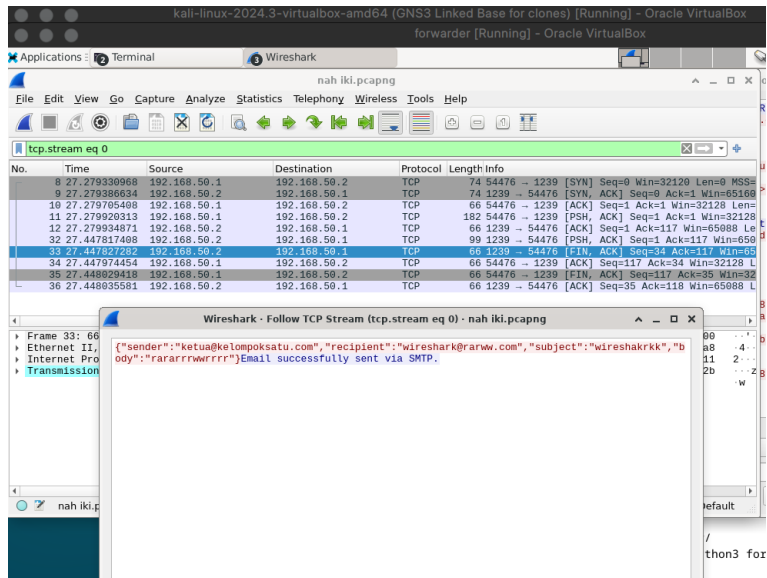


Figure 14: Socket TCP

## 3. SMTP

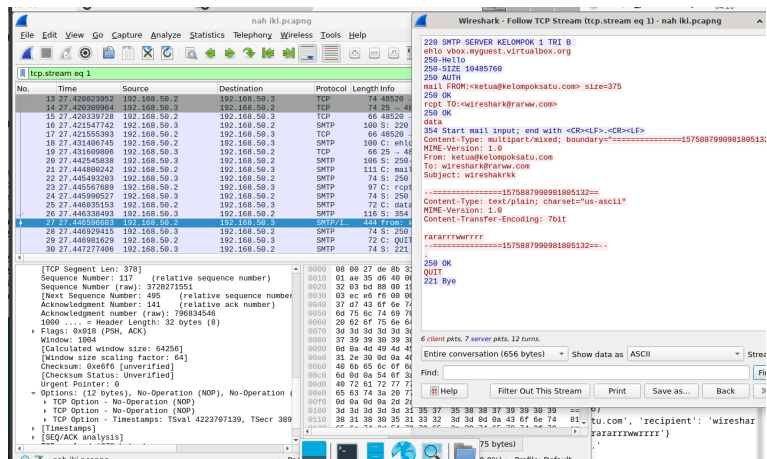


Figure 15: SMTP

## BAB IV

### ANALISA

Pada hasil percobaan yang telah dilakukan didapatkan analisa yaitu sistem ini dirancang untuk mengirimkan pesan melalui email dan menampilkannya di halaman web secara dengan menggunakan nodejs. Sistem yang dibangun terdiri dari tiga komponen utama, yaitu client, server forwarder, dan server email. Ketiganya berkomunikasi menggunakan protokol TCP untuk memastikan stabilitas dan efisiensi dalam pengiriman pesan.

Proses pengiriman dimulai ketika klien mengirimkan data email ke server forwarder melalui koneksi socket TCP. Forwarder berfungsi sebagai jembatan antara klien dan server SMTP, bertugas untuk menerima pesan dari klien, memvalidasinya, dan meneruskannya ke server SMTP. Selain itu, forwarder bertanggung jawab untuk memberikan respons kepada klien, baik ketika pesan berhasil diterima maupun ketika terjadi kegagalan. Respons ini dikirim dalam bentuk status kode, memungkinkan klien mengetahui hasil dari pengiriman pesan.

Setelah menerima data dari klien, forwarder meneruskannya ke server SMTP menggunakan protokol SMTP. Pada proyek ini, server SMTP dibangun menggunakan bahasa pemrograman Rust, dengan mengacu pada spesifikasi dalam RFC 5321 sebagai pedoman utama. RFC 5321 menyediakan standar dan aturan rinci untuk menangani perintah-perintah SMTP seperti HELO/EHLO, MAIL FROM, RCPT TO, dan DATA, serta untuk memastikan pengiriman pesan yang sesuai dengan protokol.

Namun, terdapat beberapa tantangan dalam membangun server SMTP ini. Salah satu tantangan utama adalah memastikan bahwa server SMTP dapat menangani semua perintah, status, dan respons sesuai dengan spesifikasi RFC 5321. Implementasi harus mencakup kemampuan untuk memproses berbagai status kode seperti 250 OK, 501 Syntax: Invalid email address, dan kode lainnya, yang kemudian dikirim kembali ke forwarder. Forwarder bertugas untuk meneruskan status ini ke klien sehingga klien dapat menerima umpan balik secara real-time. Selain itu juga Dari server SMTP Ini apabila berhasil maka menyimpan data dari client untuk ditampilkan ke browser dengan menggunakan web server dalam hal ini kami menggunakan node js.

# BAB V

## KESIMPULAN

### 5.1 Kesimpulan

Kesimpulan yang bisa kita ambil yaitu, sistem yang berhasil di rancang dan di implementasikan sistem pengiriman pesan melalui email dengan mengintegrasikan tiga komponen pendukung, yaitu client, forwarder, dan server email. Komunikasi antar satu sama lain menggunakan protokol TCP untuk memastikan efisiensi dan stabil nya komunikasi tersebut. Untuk server forwarder berperan sebagai penghubung antar client dan server SMTP, bertugas untuk memverifikasi, meneruskan pesan, serta memberikan respon status berhasilnya dikirim ke client. Server SMTP yang di operasikan menggunakan bahasa Rust telah di implementasikan sesuai standar RFC 5321, untuk menangani perintah-perintah protokol SMTP serta status dan respons nya dengan cepat. Selain untuk pengiriman email, sistem ini juga memungkinkan penyimpanan data email di server yang di terima dari client melalui web server yang berbasis Node.js. Walaupun terdapat tantangan teknis dalam memastikan kesesuaian implementasi dengan spesifikasi protokol SMTP, sistem ini telah mampu untuk memberikan umpan balik secara real time kepada client dan mendukung pengiriman pesan yang efisien.