

# **Tugas Besar**

## **IF2230 - Jaringan Komputer**

*"ChatTCP"*

Dipersiapkan oleh:

妹ラボラトリー

(Asisten Laboratorium Sistem Terdistribusi)

**Sister; Lab<sup>22</sup>**

**Deadline: Minggu, 1 Juni 2025 23.59 WIB**

## **1. Daftar Revisi**

---

1. Memperjelas ukuran payload di [bagian 4.1.1](#) (28 Mei, 2025, pukul 15.21).
2. Menambahkan alternatif tunneling [bagian 4.3](#) (30 Mei, 2025, pukul 11.28).

## [?] Latar Belakang

---

Bagian ini tidak wajib untuk dibaca.

*Du weißt vielleicht schon, dass dieser Kurs eigentlich nach IF2130 „Betriebssysteme“ belegt werden sollte. Und das aus gutem Grund. Betriebssysteme sind im wahrsten Sinne des Wortes Voraussetzung für diesen Kurs. Daneben scheinen auch viele andere Kurse den Schwierigkeitsgrad/Arbeitsaufwand ihrer Aufgaben zu erhöhen, vielleicht jeder aus eigenen Gründen. Das macht dein akademisches Erlebnis ziemlich einzigartig und auch sehr stressig. Falls es dir noch niemand gesagt hat: **Du machst das schon großartig, dass du es so weit geschafft hast.***

*Es war ein ziemlich hektisches Semester, nicht nur für euch, sondern auch für uns. Wusstet ihr, dass in den Stundenplänen nicht einmal Platz für die Laborarbeit für diesen Kurs war, weshalb wir sie auf zwei Sitzungen aufteilen mussten? Ja, das war nicht lustig. Die Stundenpläne für dieses Semester sind wirklich schrecklich. Eigentlich sollte die Laborarbeit in der Woche nach den Zwischenprüfungen enden. Aber wegen zwei zusätzlichen Feiertagen (die eine Woche nicht nutzen konnten) und auch wegen des verzögerten Beginns aus anderen Gründen, wurde es am Ende ziemlich spät. Wir entschuldigen uns aufrichtig, falls ihr den Stundenplan ungünstig oder unpassend findet.*



*(Ein völlig unabhängiges Bild von Tanya Degurechaff)*

*Source: [Tanya degurechaff von Misaki Nonaka](#)*

*Wie dem auch sei, wir glauben, dass wir den Schwierigkeitsgrad dieses Projekts reduziert haben und es einigermaßen in Ordnung ist. Hoffentlich ist es jetzt nicht zu schwierig. Es ist zwar immer noch recht langwierig, aber angesichts der Verwendung des berüchtigten Python und der Existenz von LLMs halten wir den Arbeitsaufwand für angemessen. Nicht, dass die Verwendung*

*von LLMs von Ihnen erwartet wird. Leider muss das Projekt einfach so gestaltet sein, dass Studierende, die das Ganze mit KI-Code ausprobieren möchten, zumindest verstehen müssen, was der von der KI bereitgestellte Code bewirkt.*

*In diesem Sinne hoffen wir, dass Sie KI sinnvoll einsetzen und dem wirklichen Verständnis des Materials Priorität einräumen. Denken Sie daran, dass es sich um ein Werkzeug handelt und Sie mehr davon haben, wenn Sie sich weiter mit den Themen befassen, anstatt einfach nach dem Code zu fragen, der Ihnen auf dem Silbertablett serviert wird. Es ist uns egal, ob Sie kopieren und einfügen. Uns ist wichtig, dass Sie sich bemühen, jeden Code zu verstehen, den Sie kopieren und einfügen.*

*Wir wünschen Ihnen viel Erfolg und alles Gute.*

*~ IF2230 Kurs Assistenten*

*P.S.: Ich verstehe eigentlich kein Deutsch. Ich finde es einfach eine coole Sprache.*

## 2. Tujuan dan Aturan

---

### 2.1 Tujuan

Tugas ini adalah bagian dari mata kuliah. Dengan menyelesaikan tugas ini, peserta diharapkan memiliki luaran berikut.

1. Memahami mekanisme dasar pengiriman data melalui jaringan.
2. Memahami peran dan tanggung jawab ***transport layer protocol***.
3. Memahami konsep-konsep pada TCP untuk memastikan *reliability* dalam jaringan yang buruk (*unreliable environment*).
4. Memahami cara kerja aplikasi *client-server* sederhana.

### 2.2 Aturan

Berikut adalah aturan-aturan yang berlaku pada pengerjaan tugas besar ini.

1. Kerjakan dan kumpulkan tugas Anda sesuai dengan arahan pengumpulan yang terdapat pada bagian [\*\*Pengerjaan dan Deliverables\*\*](#).
2. Anda **diperbolehkan** menggunakan sumber-sumber eksternal, termasuk internet, *large language model* seperti ChatGPT, serta *repository* kode yang sudah ada sebagai referensi. Namun, pahamilah seluruh kode yang Anda baca dan gunakan.
3. Diskusi antar-kelompok juga diperbolehkan, tetapi Anda tetap **dilarang** menyalin pekerjaan kelompok lain, apalagi melakukan pengerjaan untuk kelompok lain. Tolong bertanggung jawab atas pekerjaan Anda sendiri.
4. Anda sangat dilarang melakukan kecurangan atau tindakan apapun yang merugikan peserta IF2230 lain.
5. Tanyakan segala pertanyaan pada [\*\*sheets OnA\*\*](#), tetapi pastikan jawaban dari pertanyaan Anda belum terdapat di spek ataupun pertanyaan yang sudah ada.

## 3. Prerequisites

---

Tugas ini akan dikerjakan menggunakan bahasa **Python** sesuai request. Bagian ini ditujukan sebagai pengenalan beberapa *major points* pada tugas ini: *setup environment & project* di Python, *socket programming* menggunakan Python, serta *client-server application model*.

### 3.1 Setup Environment & Project

Dalam pengembangan proyek Python, penggunaan *virtual environment* sangat disarankan untuk mengelola dependensi atau *library*. Dengan *virtual environment*, setiap proyek memiliki lingkungan yang *consistent across machines* dan terisolasi sehingga tidak terjadi konflik antar proyek atau dengan sistem Python utama. Isolasi ini sangat berguna, terutama saat setiap proyek membutuhkan versi *library* (atau bahkan versi Python) yang berbeda.

Untuk mengenalkan kalian dengan *tools* yang lebih modern, pada tugas besar ini kita akan menggunakan *package/project manager* **uv**. *Project manager* ini memiliki performa lebih baik dibandingkan **pip** (yang biasa dipakai), karena **uv** menggunakan mekanisme *parallel downloads* dan *optimized dependency resolver* yang diimplementasikan menggunakan **Rust**.

Instalasi **uv** dapat dilakukan dengan mengikuti langkah pada [Installation | uv](#), pastikan untuk mengikuti langkah sesuai OS yang dipakai.

```
uv init <project-name>
```

Contoh:

```
uv init chat-tcp
```

Jalankan program dengan *command* berikut.

```
uv run main.py
```

Dengan menjalankan command di atas, **uv** akan secara otomatis membuat *virtual environment* (kalau belum ada) dan mengaktifkannya. Oleh karena itu, tidak perlu lagi mengaktifkan virtual environment secara manual karena **uv** akan mendeteksi keberadaan `.venv` secara otomatis.

Untuk tugas besar ini, kita akan menggunakan *latest stable version* dari python yaitu 3.13. Kalau Anda belum meng-*install* versi tersebut, eksekusi *command* berikut.

```
uv python install 3.13
```

Lalu, eksekusi *command* berikut untuk memastikan uv menggunakan python versi 3.13

```
uv python pin 3.13
```

**Perlu diperhatikan kalau pengubahan versi python dilakukan di terminal dalam IDE (contoh: VSCode), diperlukan *restart* IDE agar perubahannya teraplikasikan.**

Cek kesamaan versi dengan command berikut:

```
python --version
```

Pastikan output-nya:

```
Python 3.13.x
```

**Note:** *Folder virtual environment (.venv)* umumnya dimasukkan dalam *.gitignore* agar tidak membebani *repository* dengan *dependencies* yang ukurannya sangat besar. *Virtual environment* selalu dapat direplikasi selama file *pyproject.toml* masih ada.

## 3.2 Python Socket Programming

Dalam Python, pemrograman jaringan digunakan menggunakan modul `socket` yang memungkinkan koneksi antara dua perangkat melalui protokol TCP/IP atau UDP. Untuk memulai komunikasi, pertama Anda dapat membuat `socket` dengan `socket.socket()`, biasanya dengan parameter `socket.AF_INET` (untuk IPv4) dan `socket.SOCK_STREAM` (untuk TCP). Untuk UDP, gunakan `socket.SOCK_DGRAM`.

Dasar dari pengiriman dan penerimaan data menggunakan `socket` adalah dengan metode `send()` dan `recv()`. Fungsi `send()` digunakan untuk mengirim data (dalam bentuk `byte`), sedangkan `recv()` untuk menerima data dari koneksi. Contohnya sebagai berikut.

```
# Server side
import socket
host = '127.0.0.1'
```

```
port = 5000

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((host, port))
server_socket.listen(5)

conn, addr = server_socket.accept()
while True:
    data = conn.recv(1024).decode()
    if not data:
        break
    conn.send('Hello, client'.encode())
conn.close()
```

Kemudian berikut adalah kode pada client side.

```
# Client side
import socket

host = '127.0.0.1'
port = 5000

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((host, port))

client_socket.send(b'Hello, server!')
data = client_socket.recv(1024).decode()
client_socket.close()
```

**Note:** Contoh ini sangat minimal dan bukan untuk dijadikan *template*.

Namun, fungsi `recv()` dan `accept()` secara *default* bersifat **blocking**, artinya eksekusi program akan ‘terjebak’ hingga data tersedia. Untuk mencegah hal tersebut, kita bisa mengatur **timeout** agar `socket` hanya menunggu dalam jangka waktu tertentu. Jika waktu tunggu habis tanpa data diterima maka akan muncul `exception socket.timeout`. Gunakan fungsi berikut.

```
socket.settimeout(5)
```

Selain menggunakan `timeout`, Anda juga bisa mengatur `socket` menjadi **non-blocking** dengan `socket.setblocking(False)`. Dalam mode ini, jika `recv()` atau `send()` dipanggil dan tidak ada data (atau `socket` belum siap), maka Python langsung melempar

`BlockingIOError` daripada menunggu. Ini berguna untuk aplikasi yang butuh performa tinggi atau harus tetap merespons hal lain sembari menunggu data masuk. Oleh karena itu, pilih *blocking* atau *non-blocking operation* sesuai kebutuhan Anda.

Contoh:

```
# Blocking
socket.setblocking(True)
data = sock.recv(1024)

# Non-blocking
socket.setblocking(False)
try:
    data = sock.recv(1024)
except BlockingIOError:
    pass
```

Kemudian perhatikan ketika mengirim data, data tersebut tidak bisa sembarang, tetapi harus berupa bytes, yaitu sebuah kelas dalam Python yang merepresentasikan bytes mentah. Gunakan [library struct](#) untuk mengemas data kompleks menjadi bytes yang bisa dikirim dengan socket. Contoh penggunaannya yang paling sederhana seperti berikut.

```
import struct

N = 42          # int (4 byte)
F = 3.14        # float (4 byte)
msg = b'halo!'  # 5 char = 5 byte

# Format: 'i' = int, 'f' = float, '5s' = string 5 byte
packed_data = struct.pack('if5s', N, F, msg)

client_socket.send(packed_data)

# kemudian untuk unpack dari sisi penerima:
num, fl, message = struct.unpack('if5s', packed_data)
print(num, fl, message) # seharusnya muncul "42, 3.14, halo!"
```

### 3.3 Client-Server Application Model

Seperti yang sempat disinggung sebelumnya, Anda akan menggunakan *client-server application model*. **Client-Server Application Model** adalah arsitektur yang terdiri dari dua komponen utama, *client* dan *server*. Keduanya berkomunikasi untuk saling bertukar data.

Dalam model ini, ***client*** adalah pihak yang meminta layanan. Misalnya, *browser*, aplikasi *mobile*, atau sebuah program. Sementara itu, ***server*** adalah pihak yang menyediakan layanan atau data. Misalnya, *web server*, *database server*, atau API.

Dalam *client-server application model*, *client* mengirim permintaan (*request*) ke *server*, lalu *server* memprosesnya dan mengirim balasan (*respons*). Hasilnya, sistem menjadi lebih modular dan fleksibel karena *client* dan *server* bisa dikembangkan atau diubah secara terpisah selama keduanya tetap memiliki protokol komunikasi yang sama. Misalnya, HTTP, TCP, atau WebSocket.

Model ini sangat penting dalam **web-based development (WBD)** karena hampir semua aplikasi *web* menggunakan pola *client-server*. Misalnya, saat Anda membuka *website*, *browser (client)* mengirim *request* ke *web server* untuk mengambil halaman HTML. *Server* memproses permintaan itu, mungkin mengambil data dari *database*, lalu mengirimkan *respons* ke *client* untuk ditampilkan. Anda juga mungkin pernah mendengar API *server* yang dapat diakses oleh banyak *client* berbeda, *browser*, *mobile app*, dan lainnya.

## 4. Deskripsi Tugas

---

Tugas ini dibagi menjadi dua bagian utama, yaitu **implementasi fitur TCP di atas UDP** dan **pembuatan aplikasi *chat room* sederhana**.

### 4.1 Implementasi Fitur TCP di Atas UDP

Ingat bahwa ada dua protokol utama di *transport layer* dalam paket protokol internet, yaitu TCP (*Transmission Control Protocol*) dan UDP (*User Datagram Protocol*). TCP menyediakan pengiriman data yang *reliable*, terurut, dan terdeteksi kesalahannya.

Untuk mencapai hal-hal tersebut, ada beberapa mekanisme dalam implementasi TCP. Di antaranya adalah *acknowledgement*, nomor urut (*sequence number*), dan mekanisme deteksi kesalahan seperti *checksum*. Di sisi lain, UDP adalah protokol sederhana tanpa koneksi yang tidak menjamin keandalan atau pengurutan.

Anda diminta untuk menambahkan kelebihan-kelebihan yang ditawarkan TCP dengan cara **membungkus UDP Socket di Python bersama implementasi fitur TCP dalam sebuah kelas/objek**.

Agar lebih terbayang maksudnya, simak contoh *server* dan *client* menggunakan kakas UDP socket berikut.

Program server:

```
import socket

# Create and bind socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(('0.0.0.0', 41234))

# Wait for incoming messages
while True:
    data, addr = server_socket.recvfrom(1024)
    print(f"Received message: {data.decode()} from {addr[0]}:{addr[1]}")
```

Program *client*:

```
import socket

# Create socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Send message to server
client_socket.sendto(b"Hello, server!", ('127.0.0.1', 41234))

# Close socket
client_socket.close()
```

Dua program di atas menggunakan UDP. Jika dua program ini dijalankan pada jaringan yang tidak stabil, pengiriman paket akan berkemungkinan untuk gagal. Maka, kita ingin membungkus socket-socket dalam sebuah objek atau kelas sehingga kita bisa menambahkan fitur TCP. Misalnya, seperti berikut.

```
class BetterUDPSocket:
    def __init__(self, udp_socket=None):
        ...
        # Use the provided socket or create a new one
    def send(self, data):
        ...
        # Send to connected client
    def receive(self):
        ...
        # Receive from connected server
    def connect(self, ip_address, port):
        ...
        # Initiate 3-way handshake here
    def listen(self):
        ...
        # Listen for 3-way handshake here

    # Add other methods
```

**Note:** contoh ini untuk memberikan ilustrasi, boleh diikuti boleh tidak. Anda juga kemungkinan besar akan mengimplementasi beberapa kelas berbeda untuk meng-handle fitur-fitur TCP. Ini hanya untuk memberikan gambaran umum tugas saja.

Fitur-fitur yang wajib Anda implementasikan adalah sebagai berikut.

### 4.1.1 TCP Segments

Fitur yang dimaksud di sini adalah ***arbitrary data size***. UDP secara teori hanya memungkinkan pengiriman maksimal 1500 bytes untuk satu segment (sekali pengiriman satu segment) akibat *Maximum Transmission Unit*. **Bagilah data/payload menjadi beberapa segment**, masing-masing payload berukuran  $\leq 64$  bytes (**menyesuaikan dengan besar header**). Perhatikan bahwa ini adalah *arbitrary constraint*. Layanan yang di-offer oleh TCP adalah permasalahan desain algoritma. Buktikanlah bahwa Anda bisa mendesain algoritma yang mampu menangani *constraint* network seburuk apapun.

Kemudian, masing-masing *segment* harus dilengkapi dengan sebuah ***header*** yang setidaknya berisi:

#### Source dan destination port (masing-masing 16 bit)

Berturut-turut mengidentifikasi port asal dan port tujuan.

#### Sequence Number (32 bit atau 64 bit)

Angka ini adalah nomor urut data untuk segmen tertentu dari sesi saat ini. Ini membantu penerima untuk mengurutkan ulang segmen yang diterima secara tidak berurutan agar bisa melakukan rekonstruksi pesan yang benar.

Jika flag SYN di-set, maka *sequence number* adalah nomor urut awal. Nomor urut awal dihasilkan secara acak selama proses *three-way handshake* dan akan meningkat sesuai dengan jumlah data (ukuran payload, bukan ukuran seluruh segmen) pada segmen-segmen berikutnya.

Nomor yang dikonfirmasi dalam ACK yang bersangkutan adalah nomor urut saat ini ditambah 1.

#### ACK Number (32 bit atau 64 bit)

Jika flag ACK di-set, bagian ini menyimpan *sequence number* byte data selanjutnya yang diharapkan diterima oleh pengirim pesan. *Acknowledgement number* digunakan untuk tujuan pengakuan (*acknowledgement*) dan *flow control*, menandakan bahwa penerima telah berhasil menerima data hingga titik ini.

#### Checksum (16 bit)

Checksum adalah nilai numerik yang digunakan untuk memverifikasi integritas data selama transmisi. Nilai ini dihitung berdasarkan payload dan digunakan untuk mendeteksi kesalahan atau perubahan yang mungkin terjadi selama transmisi. Dengan membandingkan nilai checksum yang dihitung sebelum dan setelah transmisi, kita dapat memverifikasi integritas data.

#### Flags (minimum 3 bit)

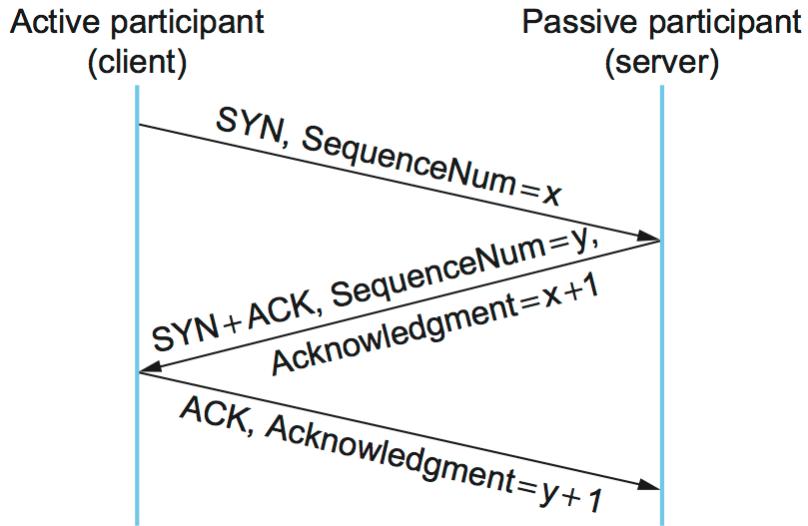
Bagian ini sebenarnya berupa **bitmap** sebesar 8 bit yang menandakan sejumlah informasi penting terkait sebuah segmen. Untuk tugas ini, Anda hanya diwajibkan untuk menyimpan SYN flag, ACK flag, dan FIN flag. Dengan kata lain, Anda cukup simpan 3 *boolean* saja. Kegunaan 3 flag/*boolean* ini akan terlihat pada penjelasan-penjelasan setelah ini.

Gunakan library struct, fungsi pack, dan unpack seperti pada [prerequisite](#) untuk mengemas header bersama payload. **Payload yang akan digunakan hanya berupa plaintext.**

Anda sangat diperbolehkan menambahkan unsur lain di *header*, misalnya jika ingin menambahkan *window size* agar jumlah *segment* pada *sliding window algorithm* bisa divariasikan (untuk melakukan *congestion control*). Atau jika Anda ingin menambahkan semua unsur *header* pada TCP yang sebenarnya, tidak masalah.

#### **4.1.2 Three-Way Handshake / Connection Establishing**

TCP menggunakan algoritma tiga langkah (*three-way handshake*) untuk memastikan bahwa kedua pihak bersedia untuk berkomunikasi satu sama lain. *Three-way handshake* ini melibatkan pertukaran tiga pesan antara klien dan server, seperti yang digambarkan pada gambar di bawah ini.



Tiga pesan dipertukarkan antara klien dan server agar kedua pihak setuju dengan aliran pesan yang akan datang dan serangkaian parameter. Proses tiga langkah ini berjalan sebagai berikut:

- Klien mengirimkan sebuah segmen ke server yang menyatakan nomor urut awal yang akan digunakan (`Flags.SYN = true, SequenceNumber = x`).
- Server merespons dengan sebuah segmen yang mengakui nomor urut klien (`Flags.ACK = true, AckNum = x + 1`) dan menyatakan nomor urut awalnya sendiri (`Flags.SYN = true, SequenceNumber = y`). Ini berarti kedua bit SYN dan ACK di-set pada bagian Flags.
- Akhirnya, klien merespons dengan sebuah segmen yang mengakui nomor urut server (`Flags.ACK = true, AckNumber = y + 1`).

Note: perhatikan penggunaan terminologi **client** dan **server** pada bagian 4.1 ini. Di sini, client dan server berturut-turut merujuk ke penerima (*receiver*) dan pengirim (*sender*) sebuah payload di transport layer, sedangkan pada bagian berikutnya (4.2), dua istilah ini merujuk ke penggunaannya yang lebih umum, yaitu pada konteks [\*\*model aplikasi client-server\*\*](#).

**Salah satu tujuan dari semua ini adalah untuk menyetujui sebuah *sequence number* awal. Namun, mengapa kita tidak hanya menggunakan 0 saja sebagai nomor urut awal?**

Ada beberapa alasan mengapa ini tidak dilakukan. Pertama, menggunakan 0 sebagai nomor urut awal dapat mengganggu koneksi TCP lainnya. Di samping itu, nomor urut yang

dapat ditebak membuat koneksi rentan terhadap pembajakan sesi. Artinya, memungkinkan penyerang untuk menyamar sebagai korban dari perspektif server.

Implementasikan algoritma *three-way handshake* untuk menghubungkan klien dan server sebelum data sebenarnya dikirim. **Anda harus menggunakan nomor urut awal yang acak untuk kedua belah pihak.** Dibebaskan untuk menentukan siapa yang akan menginisiasi *handshake*, apakah pengirim atau penerima (*manakah yang lebih mudah?*).

#### 4.1.3 Flow Control

Untuk menjaga *flow control* yang baik antara kedua pihak, TCP menggunakan algoritma *sliding window* yang berbasis SYN dan ACK. Algoritma ini memastikan bahwa paket data dikirim dan diterima dengan cara yang efisien, dengan jumlah kehilangan (*losses*) minimal dan proses retransmisi yang optimal. **Bagian ini sangat krusial karena program Anda akan diuji pada jaringan yang buruk.**

Implementasi dibebaskan, Anda bisa mengikuti skema *sliding window* yang dijelaskan pada *slide* perkuliahan, mencari referensi lain, atau bahkan menciptakan algoritma sendiri. Cukup pastikan saja pengiriman *segment* **tidak stop and wait** (i.e. *segment* yang sedang "melayang" harus bisa **lebih dari satu** pada suatu waktu). Penjelasan ini tidak berlaku untuk *three way handshake*; pada fase tersebut, client dan server **memang** harus saling menunggu.

Salah satu algoritma implementasi yang bisa dicoba adalah [ARQ Go-Back-N](#).

Agar lebih jelas lagi, **yang sangat tidak boleh adalah alur berikut terjadi secara sekuensial (ini namanya stop-and-wait):**

1. Sender mengirim segmen 1 dan menunggu tanpa melakukan apa-apa.
2. Receiver menerima segmen 1 dan mengirim ACK untuk segmen 1.
3. Sender menerima ACK untuk segmen 1.
4. Sender mengirim segmen 2 dan kembali menunggu tanpa melakukan apa-apa.
5. Dan seterusnya...

Apabila segmen payload sudah habis, implementasikan sebuah skema penutupan koneksi. Anda bisa gunakan skema [four-way handshake](#), tetapi syarat implementasi Anda adalah adanya **setidaknya satu mutual FIN-ACK**, yaitu:

1. Sender mengirim FIN ke receiver.
2. Receiver membalas dengan FIN-ACK.

Atau sebaliknya:

1. Receiver mengirim FIN ke sender.
2. Sender membalas dengan FIN-ACK.

Jika Anda ingin menginisiasi dari receiver, mekanisme agar receiver memahami bahwa itu segmen terakhir juga dibebaskan. Misalnya, Anda bisa meletakkan *termination bytes* di segmen terakhir, atau menambahkan flag pada struktur data segmen.

#### 4.1.4 Error Detection

Verifikasi integritas setiap *segment* dengan sebuah **algoritma checksum**. Proses memanfaatkan checksum terdiri dari langkah-langkah berikut:

1. Buat fungsi yang menghitung checksum berdasarkan isi segment.
2. Gunakan fungsi tersebut untuk menghitung nilai checksum dari keseluruhan segment.
3. Simpan checksum yang dihitung di header.
4. Kirim segment ke penerima yang dituju.
5. Setelah menerima segment, checksum dibaca. Fungsi checksum kemudian digunakan untuk menghitung nilai checksum baru berdasarkan segment yang diterima.
6. Nilai checksum yang diterima kemudian dibandingkan dengan checksum yang baru dihitung. Jika kedua nilai checksum cocok, ini mengonfirmasi bahwa data telah ditransmisikan dengan benar; jika tidak, ini menunjukkan adanya kesalahan dalam transmisi data.

## 4.2 Aplikasi Chat Room Sederhana

Sekarang, setelah mengimplementasi fitur-fitur TCP, waktunya Anda gunakan kelas/objek yang sudah Anda buat dalam sebuah aplikasi untuk mengirimkan data di atas sebuah jaringan. Buatlah sebuah aplikasi **chat room sederhana berbasis command line** dengan deskripsi berikut.

Aplikasi terdiri dari dua *script file*: `server.py` dan `client.py`. Singkatnya, `server.py` akan mengelola sebuah *chat room* yang bisa digunakan oleh beberapa `client.py` (dari terminal yang berbeda) untuk mengirimkan dan melihat pesan.

Agar lebih jelas, berikut adalah alur *event-event* yang terjadi. **“Pengiriman pesan” pada bagian ini merujuk ke penggunaan fungsi send dan receive buatan Anda pada bagian**

**sebelumnya.** Anda bisa anggap seluruh *behavior* kedua program yang dideskripsikan pada alur ini **wajib diimplementasi, kecuali disebutkan sebaliknya.**

1. Pertama-tama, `server.py` dijalankan pada sebuah terminal pada sebuah port. Ini yang akan menjadi application port. Mula-mula, *chat room* kosong dan *list of connected users* juga kosong.
2. Seorang user sekarang menjalankan `client.py`. User akan diminta menginput IP address dan port server yang ingin dituju, serta *display name* yang akan digunakan yang bisa dilihat oleh user lain. Anda boleh juga menerima input-input tersebut via [command line arguments](#).
3. Kemudian, server akan menerima koneksi dari client dan mendaftarkan user pada *list of connected users* (daftar ini tidak untuk ditampilkan ke user manapun, hanya data internal saja). IP Address dan *source port* masing-masing *client* juga akan diingat agar server bisa mengidentifikasi pesan-pesan yang masuk datang dari siapa saja (IP Address di sini menjadi semacam ID atau *primary key*).
4. Selama belum ditutup, `client.py` akan mengirimkan ***heartbeat message*** atau pesan berkala, misal setiap 1 detik, untuk menandakan ke *server* bahwa user tersebut masih terkoneksi. Apabila ada 30 detik berlalu tanpa *server* menerima pesan berkala dari sebuah user, user ini akan dianggap telah menutup koneksi dan *server* menghapusnya dari *list of connected users*.
5. Client akan meminta server secara berkala untuk mengirimkan data isi *chat room* untuk ditampilkan ke user. Frekuensi permintaan ini dibebaskan, misalnya dilakukan pada saat yang bersamaan dengan *heartbeat message*. Setelah mendapat data itu, Client akan me-refresh tampilan terminal pada user. **Tidak perlu tampilkan seluruh pesan, cukup tampilkan sejumlah pesan terakhir (misalnya 20 pesan terakhir).**
6. Client senantiasa meminta text input dari user. Setiap kali user mengetik *newline*, akan dianggap sebagai sebuah pesan yang akan dikirimkan ke server dan masuk ke *chat room*.
7. Server akan menerima semua pesan dari tiap-tiap user dan akan dimasukkan ke *chat room*, layaknya *group chat* pada umumnya. Data tiap pesan yang wajib ditampilkan adalah sebagai berikut.

<display_name> [<timestampl>]: <message>
--

8. Pesan khusus oleh server ditandakan dengan *display name* SERVER. User tidak diperbolehkan menggunakan *display name* tersebut. Server akan menampilkan pesan khusus ketika ada user yang bergabung atau meninggalkan *chat room*.
9. Server dapat mati sewaktu-waktu. Jika itu terjadi, client harus bisa mendeteksi dan menampilkan pesan kepada user bahwa hal tersebut terjadi, kemudian melakukan terminasi.

Contoh tampilan client sebagai berikut (boleh diikuti, boleh tidak, tetapi informasi yang harus ditampilkan minimal seperti yang ada di contoh).

```
> python client.py 192.168.1.33 -p 34112 -name User1

Connected to 192.168.1.33's chat room (4 online users)
-----
SERVER [9:13 PM] - User1 has joined!
User1 [9:13 PM]: Hello, guys!
User2 [9:13 PM]: heyy
Gurt [9:13 PM]: yo
User3 [9:13 PM]: ada yang udah mulai nyari-nyari kp kah?
User1 [9:14 PM]: lah aku udah dapet sih hehe
User2 [9:14 PM]: iya sama aku jg hehe
SERVER [9:14 PM] - User3 has disconnected!

-----
Enter text message:
```

Perhatikan ada pesan "*User3 has disconnected*". Bedakan pesan ini dengan kasus ketika User3 sudah *idle* atau terlalu lama tidak mengirimkan *heartbeat message*. Pesan dibebaskan, misalnya "*User3 was AFK and has been kicked from the chat!*".

User juga bisa mengirim pesan khusus yang diawali tanda seru (!). Pesan-pesan khusus ini adalah:

1. `!disconnect` untuk menutup koneksi. Tentunya *server* akan menghapus user dari *list of connected users*-nya.
2. `!kill <password>` untuk mematikan server secara paksa dari sisi *client*. Password boleh apa saja, yang penting ada (misalnya mau dijadikan parameter ketika menjalankan server, atau mau di-hardcode sebagai konstan saja tidak masalah).
3. `!change <name>` untuk mengubah *display name*.

## 4.3 Spesifikasi Bonus

Berikut spesifikasi bonus yang bisa dikerjakan.

1. **[+12 poin]** Lakukan *port forwarding* pada *router* Anda agar *chat room* bisa diakses oleh perangkat mana saja di dunia (tidak harus satu LAN).
2. **[+5 poin]** Jika bonus sebelumnya tidak memungkinkan (misalnya jika *router* tidak bisa diakses atau tidak mempunyai *public IP*), diperbolehkan melakukan *deployment* dengan metode lain, misalnya [ngrok](#) atau [pinggy](#). *Deployment* di *third party end-device* (misalnya di *cloud* atau *web hosting* seperti Vercel) tidak dihitung, harus di perangkat pribadi.

**Note:** Bonus ini ***mutually exclusive*** dengan bonus pertama. Jika Anda mengerjakan keduanya, maka poin tambahan Anda hanya akan didapat dari yang pertama saja.

3. **[+?? Poin]** Kreativitas implementasi.

## 5. Ketentuan Tambahan dan FAQ

---

Ketentuan tambahan akan ditambahkan di sini.

1. Tidak boleh import library di luar bawaan Python. Artinya, penggunaan *virtual environment* di sini memang hanya untuk *best practice* dan isolasi saja (juga agar yang belum pernah menggunakan menjadi tahu).
2. Komunikasi **harus** dilakukan menggunakan socket dengan protokol UDP. Pastikan inisialisasi socket dengan `type=socket.SOCK_DGRAM`. **Pemakaian tipe socket SOCK\_STREAM (TCP) akan mengakibatkan nilai tugas besar menjadi 0.**
3. MTU pada tugas ini dianggap 128 bytes sehingga pengiriman data pada socket bawaan Python **tidak diperbolehkan melebihi 128 bytes**: maksimal `socket.recvfrom(128)` dan ini pun untuk satu segmen. Alokasikan maksimal 64 byte untuk header dan sisanya untuk payload. **Pengiriman lebih dari 128 bytes juga akan mengakibatkan nilai tugas besar menjadi 0.**
4. Kreativitas dalam penggeraan sangat dianjurkan untuk memperdalam pemahaman. **Penilaian sepenuhnya didasarkan dari kriteria penilaian**, bukan detail implementasi.
5. Tampilan terminal chatroom **tidak harus bisa di-scroll**. Pengguna memang tidak harus bisa melihat seluruh *history chat* di terminal. Ketika ada pengguna yang baru join, dibebaskan apakah ingin menampilkan chat yang sudah ada atau tidak.
6. Ketika pengujian, Anda harus bisa menjalankan minimal 2 client dan 1 server, masing-masing dari host yang berbeda (di jaringan yang sama) agar IP *address*-nya berbeda juga. Anda bisa menggunakan VM, atau gunakan beberapa perangkat.
7. Dari poin sebelumnya, jika Anda mengerjakan bonus, maka server harus bisa berada pada jaringan yang berbeda dengan para client.

## 6. Pengerjaan & Deliverables

---

1. Tugas dikerjakan berkelompok dengan 4-5 orang. Kelompok **tidak lintas kelas**. Isi kelompok kalian pada [tautan sheets ini](#). Tenggat waktu pengisian sheets adalah Minggu, 18 Mei 2025 pukul 21.00. Setelah waktu tersebut, sheets akan dikunci dan peserta mata kuliah yang belum mendapat kelompok akan di-*assign* secara acak ke kelompok yang sudah ada atau kolom kelompok yang kosong.
2. Untuk tugas ini Anda diwajibkan menggunakan version control system git dengan menggunakan sebuah repository private di Github Classroom yang disediakan. Gunakan [link assignment ini](#) untuk membuat repository.
3. Pengumpulan dilakukan dengan membuat **release** dengan tag `v1.0` pada repository yang telah kelompok Anda buat. Jika terdapat revisi, tambahkan angka minor pada versi tag (`v1.1`, `v1.2`, ..., `v1.x`). Pastikan tag sesuai format. Repository team yang tidak memiliki tag ini akan dianggap tidak mengumpulkan.
4. Repository **minimal** berisi beberapa hal berikut.
  - a. Source code program.
  - b. `README.md`, yang minimal berisi deskripsi singkat repository, cara setup dan run program, pembagian tugas tiap anggota kelompok, dan semua referensi yang digunakan (jika ada).
  - c. `LICENSE`, yang berisi lisensi kode program *supaya kerennya*.
5. Penilaian akan dilakukan pada *release* terakhir dengan tag `v1.x`, **bahkan jika release atau pengumpulan form tersebut melewati deadline**, jadi hati-hati.
6. Batas akhir pengumpulan adalah hari **Minggu, 1 Juni 2025 23.59 WIB**.
7. Tugas yang terlambat dikumpulkan tetap akan diterima dengan pengurangan nilai yang proporsional terhadap tingkat keterlambatan. Namun, pengumpulan atau rilis lewat dari 24 jam setelah *deadline* tidak diterima dan otomatis diberi nilai 0.

# 7. Penilaian & Simulasi Jaringan Buruk

---

## 7.1 Kriteria Penilaian

Program akan dinilai secara kelompok. Untuk nilai demo akan dinilai secara individu (informasi terkait demo akan diberitahukan pada kemudian hari). Anda bisa asumsikan seluruh pengujian akan dilakukan pada simulasi jaringan yang buruk. Berikut poin-poin penilaian beserta nilai maksimumnya.

- |  |             |
|--|-------------|
| 1. Fitur pada protokol ChatTCP sesuai spesifikasi. | <b>(30)</b> |
| a. TCP Segments                                    | (7.5)       |
| b. Three-Way Handshake                             | (7.5)       |
| c. Flow Control                                    | (7.5)       |
| d. Error Detection                                 | (7.5)       |
| 2. Aplikasi chat room berjalan sesuai spesifikasi. | <b>(30)</b> |
| a. Fitur mengirim dan membaca pesan biasa          | (20)        |
| b. Fitur mengirim dan mengeksekusi pesan khusus    | (10)        |
| 3. Demo  | <b>(40)</b> |
| 4. Bonus   | <b>(12)</b> |
| 5. Kreativitas Implementasi                        | (??)        |

Untuk penggeraan bonus akan diuji saat demo.

## 7.2 Cara Simulasi Jaringan yang Buruk

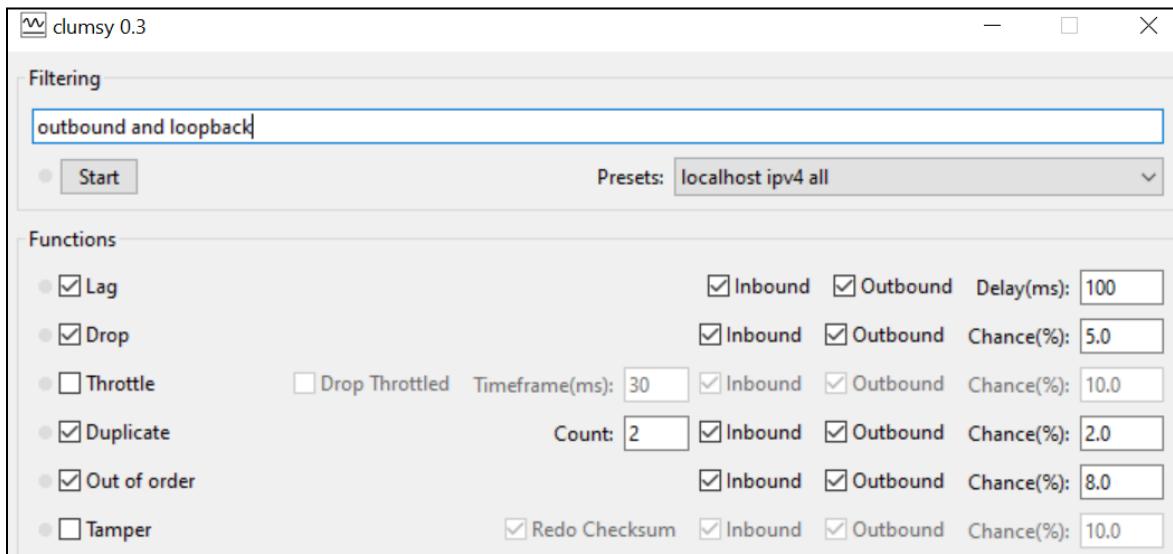
Untuk linux, jalankan perintah ini di terminal Anda dengan akses superuser. Disarankan untuk menggunakan VM Ubuntu untuk memudahkan. Bagian ini akan digunakan untuk mensimulasikan jaringan buruk.

```
$ tc qdisc add dev lo root netem delay 100ms 50ms reorder 8% corrupt  
5% duplicate 2% 5% loss 5%
```

**Penting:** setelah selesai, lakukan perintah ini untuk mengembalikan konfigurasi network interface yang diubah untuk keperluan demo:

```
$ tc qdisc del dev lo root netem delay 100ms 50ms reorder 8% corrupt  
5% duplicate 2% 5% loss 5%
```

Untuk Windows, substitusi command `tc` dengan <https://jagt.github.io/clumsy/>. Unduh `clumsy-0.3-win64-a.zip`, extract kemudian jalankan `clumsy.exe`. Lalu, ikuti konfigurasi berikut.



Tekan tombol start, dan jangan lupa untuk mematikan ketika sudah selesai.

Selain *tools* di atas, Anda juga bisa gunakan VM atau WSL untuk menjalankan program di lingkungan Linux.

## Pesan Asisten

~ guys ayo balik ke semester 1 aja yuk😭aku sdh lelah ~

Aldy

~Chimpanzini Bananini! Wa wa wa! Bananuchi monkey monkey monkey uchi!~

Albert

~ Jaga kewarasan kalian ya guys ~

Onta



Tubesnya udah dipermudah yak, sekarang info loker 😊 ~

Ryle

~



jangan lupa asistensi OS, sejauh ini baru 8 kelompok yang asistensi ~  
**Layla**

~ Kalo vibe coding trus pas demo ngangong, ga lulus ya ~

Owen

~



Tau ah gua bingung ngasih pesan apa. Nvm, OS-nya jangan lupa ya <3

Jarkom cuman sampingan ajah

~

Indra



Heeee?

I smelled skill issue \*sniff sniff

Tubes IF TCP pake Python? قبأي ءالاء ربكما تكبان?

Prompting 2-3x chatroom kelar ^u^

(Real Challenge is when integrating with your own TCP) ~

WazeAzure



Pol

~

Jar jar jar jar jar jar jaringan komputer

Hiiiiii seremnyaaaaaa

Sebuah entitas digital bernama Jarkom yang dipercaya menghantui mahasiswa IF semester 4 setiap kali mereka tidak paham cara kerja three-way handshake.

Konon katanya, kalau kamu salah menghitung IP address lebih dari tiga kali, jarkom akan muncul dari kabel LAN dan membisikkan

"ping... ping... timeout..." berulang-ulang

Korban entitas ini hanya dapat menatapi layar yang bertuliskan

Received message: ÅfÅ¼Å¢â,-Å"Å,Å¥Å£ÆÅ·Å©Å>Å»ÅSÅ¬Å„Å£â€

sambil berkeringat dingin

Ucapkan mantra socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sambil mengetik IP agar terhindar dari entitas ini.

~

Flora

**Thread** X

iamkirkbater and jkjustjoshing

 **iamkirkbater** Aug 23rd, 2017 at 9:37 AM  
in #www

Do you want to hear a joke about TCP/IP?

 7

7 replies

 **jkjustjoshing** 5 months ago  
Yes, I'd like to hear a joke about TCP/IP

 **iamkirkbater** 5 months ago  
Are you ready to hear the joke about TCP/IP?

 **jkjustjoshing** 5 months ago  
I am ready to hear the joke about TCP/IP

 **iamkirkbater** 5 months ago  
Here is a joke about TCP/IP.

 **iamkirkbater** 5 months ago  
Did you receive the joke about TCP/IP?

 **jkjustjoshing** 5 months ago  
I have received the joke about TCP/IP.

 **iamkirkbater** 5 months ago  
Excellent. You have received the joke about TCP/IP. Goodbye.

~ ~  
Rafiki