

Bismillah ya, dibantu Gemini lagi :"D

## I. Jaringan Komputer



### Wireshark

App

<https://youtube.com/@samekosaba>

1. **(2.5 poin)** Terdapat beberapa metode untuk melakukan *digital encoding*, yaitu perubahan informasi dalam bentuk bit menjadi bentuk sinyal yang dapat disalurkan dalam jaringan, masing-masing dengan kelebihan dan kekurangannya.

Untuk masing-masing metode berikut, jelaskan bagaimana skema *encoding* bekerja, apa kelebihannya dibanding skema sebelumnya, dan apa kekurangan baru yang diperkenalkan skema tersebut.

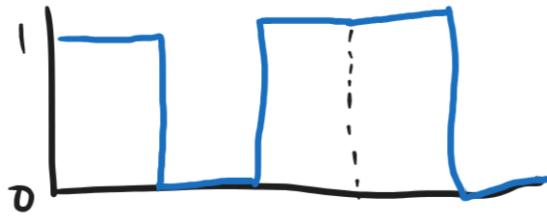
- a. **(0.5 poin)** NRZ (*Non-return to zero*)

Bit '1' direpresentasikan oleh satu level tegangan (misalnya, tegangan positif) dan bit '0' oleh level tegangan yang lain (misalnya, tegangan negatif atau nol). Sesuai namanya, sinyal tidak pernah kembali ke level nol di tengah-tengah periode bit.

- **Contoh (untuk bit stream 10110)**

Sinyal akan berada di level tinggi untuk '1', turun ke level rendah untuk '0', kembali ke level tinggi dan bertahan selama dua periode bit untuk '11', lalu turun lagi ke level rendah untuk '0'.

## NRZ



- **Kelebihan:**

- Gampang implementasi dengan sirkuit sedikit.
- Bandwidth efisien karena satu level sinyal mewakili satu bit penuh.

- **Kekurangan:**

- Jika ada string panjang bit yang sama (misalnya, 11111111), sinyalnya bakal jadi tegangan konstan untuk waktu yang lama -> ngebikin komponen DC (*Direct Current*) yang dapat ngebuat distorsi sinyal dan sulit ditransmisikan melalui beberapa media jaringan.
- String panjang bit yang sama tidak menghasilkan transisi sinyal. Tanpa transisi, jam (*clock*) di sisi penerima bisa kehilangan sinkronisasi dengan pengirim. Akibatnya, penerima bisa salah membaca jumlah bit yang diterima, sebuah masalah yang dikenal sebagai *bit slips*.

b. (0.5 poin) NRZI (Non-return to zero inverted)

Yang dilihat disini tuh perubahan tegangannya.

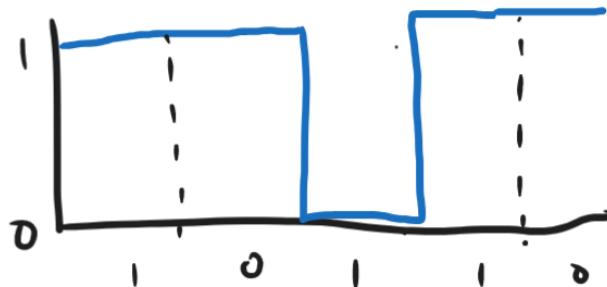
Bit '1' = ada **transisi** (perubahan level tegangan) di awal periode bit

Bit '0' = **tidak adanya transisi**

- **Contoh (untuk bit stream 10110):** Asumsi sinyal dimulai dari level rendah. Bit pertama '1' menyebabkan transisi ke tinggi. Bit kedua '0'

tidak menyebabkan transisi, sinyal tetap tinggi. Bit ketiga '1' menyebabkan transisi ke rendah. Bit keempat '1' menyebabkan transisi lagi ke tinggi. Bit terakhir '0' tidak menyebabkan transisi, sinyal tetap tinggi.

## NRZ-I



- **Kelebihan (dibanding NRZ):**

- Skema ini memecahkan masalah string panjang bit '1'. Setiap bit '1' sekarang menjamin adanya transisi, yang membantu penerima menjaga sinkronisasi jam
- Karena hanya mendekripsi perubahan, bukan level absolut, NRZI lebih tahan terhadap kesalahan waktu minor.

- **Kekurangan :** NRZI ga menyelesaikan masalah sinkronisasi, dia cuma mindahinnya. Sekarang, string panjang bit '0' yang menjadi masalah baru, karena tidak menghasilkan transisi sama sekali dan dapat menyebabkan hilangnya sinkronisasi.

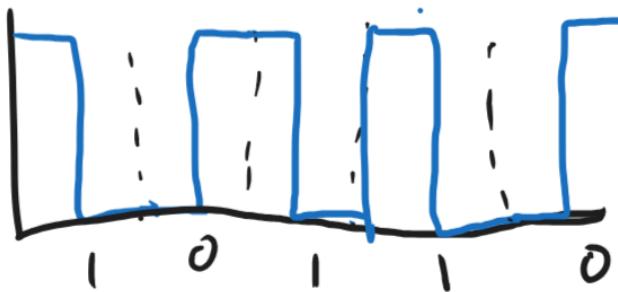
c. (0.5 poin) Manchester

Yang diliat disini jadi tuh adanya transisi di tengah-tengah setiap periode bit. Transisi ini fungsinya dua: sebagai sinyal jam dan sebagai pembawa data. Bit '0' itu dari transisi dari level rendah ke tinggi, dan bit '1' oleh transisi dari tinggi ke rendah.

- Contoh (untuk bit stream 10110): Untuk '1', sinyal akan tinggi di paruh pertama bit dan rendah di paruh kedua. Untuk '0', sinyal akan rendah

di paruh pertama dan tinggi di paruh kedua.

## Manchester



- **Kelebihan (dibanding NRZ):**

- Karena selalu ada transisi di tengah setiap bit, sinyal ini menjadi self-clocking. Penerima dapat mengekstrak sinyal jam langsung dari aliran data, yang secara efektif menghilangkan risiko bit slips.
- Setiap bit memiliki komponen tegangan positif dan negatif yang seimbang, jadi bakal gaada komponen DC.

- **Kekurangan : Boros Bandwidthhhhh.** Karena membutuhkan setidaknya satu transisi per bit (bahkan bisa dua jika bit berturut-turut sama), data rate efektifnya cuma setengah dari NRZ untuk bandwidth yang sama.

d. (0.5 poin) Differential Manchester

Ya sama kayak manchester tadi, tapi kembali lagi, yang diliat itu **transisi-nya**. Nilai bit ('0' atau '1') ditentukan oleh **ada atau tidaknya transisi di awal periode bit**. Bit '0' direpresentasikan dengan adanya transisi di awal periode, sedangkan bit '1' direpresentasikan dengan tidak adanya transisi di awal

- **Contoh (untuk bit stream 10110):** Asumsi sinyal berakhir di level tinggi. Bit pertama '1' tidak memiliki transisi di awal, jadi sinyal tetap

tinggi lalu turun di tengah. Bit kedua '0' memiliki transisi di awal, jadi sinyal turun ke rendah lalu naik di tengah. Bit ketiga '1' tidak ada transisi awal, sinyal tetap tinggi lalu turun di tengah. Dan seterusnya.

- **Kelebihan :** Keunggulan utamanya adalah kekebalannya terhadap polaritas sinyal yang terbalik (misalnya, jika kabel positif dan negatif tertukar). Karena data direpresentasikan oleh ada atau tidaknya perubahan (diferensial), bukan oleh arah transisi absolut (*low-to-high*), sinyal yang terbalik tidak akan memengaruhi interpretasi data.
- **Kekurangan: Tetap Boros Bandwidth.** Sama seperti Manchester, skema ini tetap membutuhkan *bandwidth* dua kali lipat dari NRZ, sehingga tetap tidak efisien

e. (0.5 poin) 4B/5B

Idenya adalah mengambil setiap 4 bit data (*nibble*) dan menggantinya dengan 5 bit kode yang sudah ditentukan dalam sebuah tabel. Kode 5-bit ini dipilih secara khusus untuk memastikan tidak akan pernah ada string panjang bit '0' yang dapat menyebabkan hilangnya sinkronisasi saat dikirim menggunakan NRZI

- **Contoh (untuk bit stream 0000 1111):** Nibble 0000 akan diubah menjadi kode 5-bit 11110. Nibble 1111 akan diubah menjadi 11101. Hasilnya, yang dikirim adalah 1111011101.
- **Kelebihan (dibanding skema sebelumnya):**
  - Dengan menambahkan satu bit ekstra, 4B/5B menjamin ada cukup transisi dalam sinyal untuk menjaga sinkronisasi *clock* saat digunakan bersama NRZI. Ini memecahkan masalah NRZI tanpa harus mengorbankan *bandwidth* sebesar 50% seperti pada skema Manchester
  - Dari 32 kemungkinan kode 5-bit (25), hanya 16 yang digunakan untuk data. Sisanya dapat digunakan untuk sinyal kontrol (misalnya, *start/end of frame*) atau dibiarkan tidak

terpakai. Jika penerima mendapatkan kode 5-bit yang tidak valid, ia tahu bahwa telah terjadi kesalahan transmisi

- **Kekurangan Baru:**

- **Overhead:** Skema ini memperkenalkan overhead sebesar 25% (karena 4 bit data menjadi 5 bit transmisi). Ini berarti **baud rate** (jumlah simbol per detik di media) menjadi 20% lebih tinggi dari **bit rate** (data asli).
- **Tidak Menghilangkan Komponen DC:** Skema ini tidak dirancang untuk menyeimbangkan jumlah '1' dan '0', sehingga masalah komponen DC pada NRZI masih bisa terjadi

**Referensi:** <https://en.wikipedia.org/wiki/4B5B> (baru denger ini tbh)

**Bonus:** (0.1 poin/subsoal) Berikan contoh untuk masing-masing metode.

**Bonus<sup>2</sup>:** (0.4 poin/subsoal) Berikan visualisasi untuk masing-masing contoh. Visualisasi dibuat sendiri.

2. (2.5 poin) Jawablah pertanyaan-pertanyaan terkait transmisi pada data *link layer* berikut.
  - (1 poin) Pada protokol Ethernet, terdapat sebuah masalah yang sangat umum terjadi, yaitu ketika terjadi *collision* pada transmisi data. Mengapa fenomena ini menjadi masalah? Lalu jelaskan cara kerja algoritma yang digunakan protokol Ethernet untuk mengatasi masalah tersebut!

Pada jaringan Ethernet lawas yang menggunakan media bersama (shared medium) seperti kabel coaxial atau hub, semua perangkat berada dalam satu "domain tabrakan" (collision domain). Ini berarti jika dua atau lebih perangkat mencoba mengirim data pada saat yang bersamaan, sinyal listrik mereka akan saling mengganggu di dalam kabel.

Untuk mengatasi masalah ini, protokol Ethernet menggunakan algoritma yang disebut Carrier-Sense Multiple Access with Collision Detection (CSMA/CD). Kita pakai permisalan diskusi kelompok

- **Carrier Sense**

Sebelum "berbicara" (mengirim data), setiap perangkat akan "mendengarkan" kabel untuk memastikan tidak ada perangkat lain yang sedang mengirim. Jika media sedang sibuk, perangkat akan menunggu hingga sepi

- **Multiple Access**

Semua perangkat memiliki hak yang sama untuk menggunakan media transmisi.

- **Collision Detection**

Saat sebuah perangkat mulai mengirim data, ia tidak berhenti mendengarkan. Ia terus memonitor level tegangan di kabel. Jika level tegangan yang terdeteksi tidak sesuai dengan yang sedang dikirimkannya, itu berarti ada perangkat lain yang juga mengirim data pada saat yang sama, dan terjadilah *collision*.

- **Jam Signal**

Begitu *collision* terdeteksi, perangkat akan segera berhenti mengirim datanya dan malah mengirimkan sinyal khusus yang disebut "**Jam Signal**". Sinyal ini bertujuan untuk memastikan semua perangkat lain di jaringan menyadari bahwa telah terjadi *collision*.

- **Binary Exponential Backoff (Mundur Secara Acak)**

Setelah mengirim Jam Signal, setiap perangkat yang terlibat dalam *collision* akan menunggu selama periode waktu yang acak sebelum mencoba mengirim lagi.

Waktu tunggu tidak hanya acak, tetapi rentangnya akan digandakan setiap kali *collision* terjadi secara berturut-turut untuk paket yang sama.

- b. (0.5 poin) Mengapa protokol jaringan wireless tidak bisa menggunakan metode yang sama dengan Ethernet untuk mengatasi masalah *collision* tersebut?

Karena bisa aja ada *hidden node*. Misal A-B dan B-C, keduanya bisa berkomunikasi. Nah, posisikan A-B lagi komunikasi, karena C gabisya lihat A, jadinya dia mikir B aman untuk disamper. Padahal engga. Deteksi

*collisionnya jadi susah banget di sini.*

- c. (1 poin) Jelaskan bagaimana skema alternatif yang digunakan jaringan wireless bekerja untuk mengatasi terjadinya collision.

Basically mirip, tapi pakai avoidance, bukan detection. Jadinya CSMA/CA.

1. **Listen Before Talk (LBT)**: Sama seperti awalan CSMA/CD.
2. **Random Backoff Timer**: Jika channel terdeteksi bebas, device gak langsung mengirim. Sebaliknya, ia akan memulai *timer* dengan durasi acak yang singkat. Perangkat akan terus memonitor channel selama *timer* ini berjalan. Jika channel tetap bebas sampai *timer* mencapai nol, barulah perangkat mulai ngirim. Ini secara signifikan mengurangi kemungkinan dua perangkat yang melihat channel bebas pada saat yang sama akan mulai mengirim bersamaan
3. **Mekanisme RTS/CTS**: Buat ngatasin *hidden node problem*, terutama untuk transmisi data berukuran besar, CSMA/CA pakai mekanisme **Request to Send (RTS)** dan **Clear to Send (CTS)**.
  - o Pengirim (misalnya, stasiun A) akan mengirimkan paket kecil **RTS** ke Access Point (B), yang berisi informasi tentang berapa lama ia akan membutuhkan media.
  - o Jika B siap menerima, ia akan merespons dengan paket **CTS**. Paket CTS ini akan disiarkan ke semua perangkat dalam jangkauannya.
  - o Perangkat lain (seperti stasiun C, si *hidden node*) yang mendengar paket CTS ini akan tahu bahwa media akan segera digunakan. Mereka akan mengatur **Network Allocation Vector (NAV)** mereka, yaitu sebuah *timer* internal, untuk tidak mencoba mengirim selama durasi yang ditentukan dalam paket CTS.
4. **Acknowledgement (ACK)**: Karena transmisi nirkabel rentan terhadap gangguan, setiap paket data yang berhasil diterima harus dikonfirmasi oleh penerima dengan mengirimkan paket **Acknowledgement (ACK)**. Jika pengirim tidak menerima ACK dalam jangka waktu tertentu, diasumsikan paket tersebut hilang

(mungkin karena *collision* atau interferensi) dan akan mencoba mengirimkannya kembali

**Bonus:** Jawaban dengan lebih detail terkait algoritma yang digunakan berpotensi diberikan poin tambahan.

3. (3 poin) Jawablah pertanyaan-pertanyaan terkait transmisi pada *transport layer* berikut.

- a. (1.5 poin) Apa saja informasi yang dibutuhkan untuk menentukan secara unik alamat tujuan pengiriman paket pada transmisi data *End-to-End*? Kaitkan dengan penjelasan mengapa protokol UDP disebut sebagai “*Simple Demultiplexer*” (Read: [Computer Networks: A Systems Approach, Chapter 5: End-to-End Protocols](#))

Perlu dua hal ini:

- **Alamat IP (Sumber dan Tujuan)** -> Buat Network Layer (Layer 3).
- **Nomor Port (Sumber dan Tujuan)** -> Buat Transport Layer (Layer 4).

Kombinasi dari **Alamat IP dan Nomor Port** disebut sebuah **Socket**. Jadi, untuk sebuah komunikasi *end-to-end* yang lengkap dan unik, *transport layer* butuh informasi socket sumber dan socket tujuan

- UDP disebut "simple demultiplexer" karena tugasnya sangat sederhana. Saat sebuah datagram UDP tiba, UDP hanya melihat satu hal: **nomor port tujuan** yang ada di header-nya. Berdasarkan nomor port tersebut, ia langsung meneruskan payload data ke **socket** aplikasi yang sedang "mendengarkan" di port itu.
- UDP tidak peduli dengan urutan paket, tidak memeriksa apakah ada koneksi yang sudah terjalin, dan tidak menjamin pengiriman. Ia hanya bertindak seperti penyortir surat yang sangat cepat yang hanya melihat nomor tujuan tanpa tugas tambahan apa pun

- b. (1 poin) Jika alamat IP dibutuhkan untuk menentukan tujuan pada protokol *End-to-End*, mengapa format paket pada protokol *transport layer* seperti TCP dan UDP tidak menyimpan informasi IP tujuan? Kaitkan dengan proses

enkapsulasi-dekapsulasi dan relasi *transport layer* dengan lapisan-lapisan lainnya pada OSI model.

Karena alamat IP ada di layer 3. Dari layer tinggi ke rendah akan selalu terjadi proses enkapsulasi, yakni pembungkusan dengan header dari suatu layer. Di sini, karena Port jadi tanggung jawab Transport Layer (layer 4), headernya nyimpen port. Nanti, ketika dibungkus Network Layer (layer 3), baru ada IP disitu, oleh ICMP. Jadi ga perlu ada IP di header TCP/UDP.

- c. (0.5 poin) Pada perhitungan checksum TCP, data yang digunakan bukan hanya *payload*, melainkan ditambahkan dengan sebuah *pseudo-header* yang berisi *header* dari paket TCP tersebut, serta beberapa informasi tambahan. Menurutmu, apakah penambahan data-data tambahan ini penting atau tidak untuk kebutuhan *error-detection* pada protokol TCP? Jelaskan alasannya.

**Penting.** Checksum di dalam *header* TCP dibuat untuk mendeteksi korupsi data (bit yang berubah) di dalam segmen TCP itu sendiri. Sementara itu, checksum di *header* IP (pada IPv4) hanya melindungi *header* IP, bukan data di dalamnya. Ada satu skenario berbahaya yang bisa lolos dari kedua checksum ini: **sebuah paket yang dikirim ke alamat IP yang salah (misrouted packet) tetapi isinya (segmen TCP) sama sekali tidak korup.**

- Dalam kasus ini, checksum IP akan valid karena *header* IP tidak rusak.
- Checksum TCP juga akan valid karena segmen TCP di dalamnya tidak rusak.
- Namun, paket tersebut telah sampai di mesin yang salah, dan jika diproses, bisa menyebabkan masalah serius.

Untuk mencegah hal ini, sebelum menghitung checksum, *transport layer* (baik TCP maupun UDP) secara virtual membuat sebuah "**pseudo-header**". Pseudo-header ini tidak pernah dikirimkan melalui jaringan; ia hanya ada untuk keperluan perhitungan.

Dengan memasukkan alamat IP sumber dan tujuan ke dalam perhitungan checksum TCP, TCP secara efektif mengikat segmennya ke alamat IP tujuan yang benar.

4. (4 poin) Bayangkan seorang karyawan JNE yang sedang mengantar paket yang biasanya naik tangga satu persatu. Dia harus membuka gerbang, lalu masuk ke rumah, dan akhirnya baru menyerahkan paket. Tapi sekarang, dia bisa melakukan teleportasi dan langsung meletakkan paketnya di balkon tanpa melewati prosedur panjang tersebut. Cepat, tetapi juga membuat penjaga rumah bingung karena tidak sempat mengenali siapa dia, darimana asalnya, atau apakah dia memang berhak masuk.
- Bagaimana "kurir" tersebut mencerminkan pendekatan QUIC dalam menyatukan transport dan enkripsi?
  - Apa risiko keamanan dari kurir yang tidak melewati gerbang terlebih dahulu, tetapi langsung muncul dan meletakkan paketnya?
  - Jika kurir tersebut bisa pindah dari rumah ke rumah tanpa kembali ke kantor pusat terlebih dahulu, bahkan saat pindah kota, apa tantangan dan keuntungannya dalam QUIC?

a. Kurir JNE tradisional adalah cerminan dari TCP + TLS. Ada beberapa langkah sekuensial yang memakan waktu (round-trip time atau RTT) sebelum data bisa dikirim:

- **Membuka Gerbang (TCP Handshake):** Client dan server harus melakukan "jabat tangan" tiga langkah -> 1 RTT
- **Masuk Rumah (TLS Handshake):** Setelah koneksi TCP terbentuk, mereka harus melakukan "jabat tangan" lagi untuk negosiasi enkripsi (TLS). Ini bisa memakan 1–2 RTT lagi.
- **Menyerahkan Paket:** Baru setelah itu data aplikasi bisa dikirim.

Kurir teleportasi adalah cerminan dari **QUIC**. QUIC, yang berjalan di atas UDP (protokol yang tidak butuh handshake), menggabungkan proses pembuatan koneksi dan negosiasi enkripsi menjadi satu langkah. Pada paket pertama yang dikirim *client*, ia sudah menyertakan parameter kriptografi. Ini secara drastis memotong jumlah RTT yang dibutuhkan dari 2–3 RTT menjadi hanya **1 RTT** untuk koneksi baru. Untuk koneksi yang sudah pernah terjalin, QUIC bahkan bisa mencapai **0–RTT**, di mana *client* bisa langsung mengirim data aplikasi di paket pertamanya. Inilah "teleportasi" yang memangkas latensi secara signifikan.

- b. Risiko utamanya adalah serangan replay (replay attack).

- Dalam analogi, seorang penjahat bisa merekam "kemunculan" kurir teleportasi ini dan "memutarnya ulang" berkali-kali, menyebabkan paket yang sama diletakkan di balkon berulang kali tanpa sepengetahuan penjaga rumah
- Dalam konteks QUIC, fitur 0-RTT berarti data aplikasi dikirim sebelum handshake kriptografi selesai sepenuhnya. Seorang penyerang di tengah jaringan (*man-in-the-middle*) dapat menangkap paket 0-RTT yang terenkripsi ini dan mengirimkannya kembali ke server berkali-kali.
- Karena server belum memiliki konteks penuh dari koneksi yang baru, ia mungkin akan memproses data yang sama berulang kali
- Konsekuensinya bisa bahaya jika data yang dikirim bersifat non-idempoten (aksi yang tidak boleh diulang), misalnya:
  - Permintaan "tambahkan barang X ke keranjang belanja" bisa menyebabkan barang X ditambahkan berkali-kali.
  - Permintaan "transfer \$10 ke rekening Y" bisa menyebabkan transfer terjadi berulang kali.

c. Tantangan dan keuntungan:

• Keuntungan:

Dalam TCP, sebuah koneksi diidentifikasi secara kaku oleh 4-tuple: (IP sumber, port sumber, IP tujuan, port tujuan). Jika seorang pengguna berpindah jaringan (misalnya, laptopnya beralih dari Wi-Fi kantor ke jaringan seluler 4G), alamat IP-nya akan berubah. Akibatnya, koneksi TCP akan putus dan harus dibuat ulang dari awal. Ini sangat mengganggu untuk aplikasi real-time seperti streaming video atau panggilan suara.

QUIC memecahkan masalah ini dengan menggunakan **Connection ID (CID)**, sebuah pengenal unik yang tidak terikat pada alamat IP atau port. Dengan CID, "kurir" (koneksi QUIC) dapat dengan mulus "pindah rumah" (berpindah dari jaringan Wi-Fi ke 4G) tanpa harus "kembali ke kantor pusat" (memulai koneksi baru). Koneksi tetap berjalan tanpa gangguan, memberikan pengalaman pengguna yang jauh lebih superior, terutama di perangkat mobile.

• Tantangan:

- **Validasi Jalur (Path Validation):** Sebelum koneksi bisa pindah ke alamat IP baru, QUIC harus memastikan bahwa alamat baru tersebut benar-benar milik client dan dapat dijangkau. Ini dilakukan melalui mekanisme PATH\_CHALLENGE dan

*PATH\_RESPONSE* untuk mencegah serangan di mana penyerang mencoba membajak koneksi ke alamat mereka.

- **NAT Rebinding:** Perangkat di belakang NAT (Network Address Translation) bisa saja alamat IP dan port publiknya diubah oleh router secara tiba-tiba. TCP tidak bisa menangani ini dan koneksi akan putus. QUIC dirancang untuk dapat pulih dari *NAT rebinding* dengan memperlakukannya sebagai sebuah migrasi koneksi.
- **Visibilitas dan Manajemen Jaringan:** Karena hampir seluruh header QUIC (termasuk informasi yang biasanya terlihat seperti nomor port) dienkripsi, perangkat jaringan perantara seperti *firewall* perusahaan dan sistem monitoring menjadi "buta". Mereka tidak bisa lagi dengan mudah menginspeksi, memprioritaskan, atau memblokir lalu lintas berdasarkan port seperti yang biasa mereka lakukan pada TCP. Ini menjadi tantangan besar bagi administrator jaringan yang perlu mengelola dan mengamankan jaringan mereka.

Referensi : <https://www.rfc-editor.org/rfc/rfc9001.html>

5. (2.5 poin) Kamu adalah seorang engineer yang sedang mengembangkan aplikasi kesehatan bernama SATUSEHAT. Saat melakukan evaluasi arsitektur sistem, kamu menyadari bahwa komunikasi antara client dan server masih berlangsung secara langsung, tanpa menggunakan reverse proxy. Menurutmu, apakah perlu menambahkan reverse proxy dalam arsitektur sistem ini? Jelaskan pendapatmu dengan mempertimbangkan aspek keamanan, skalabilitas, performa, dan kemudahan pengelolaan. Sampaikan argumenmu seolah-olah kamu sedang menjelaskan kepada para stakeholder non-teknis yang perlu memahami manfaatnya dalam konteks bisnis dan operasional.

Bapak dan Ibu sekalian, saat ini, aplikasi SATUSEHAT kita seperti sebuah toko yang pintunya langsung menghadap ke jalan raya yang ramai. Siapa pun bisa langsung datang dan berinteraksi dengan kasir (server aplikasi kita). Ini memang terlihat sederhana, tapi sangat berisiko dan tidak efisien.

Apa yang kita usulkan adalah menempatkan seorang **bodyguard profesional (yaitu, reverse proxy)** di depan toko kita. Jadi, semua pengunjung harus melewati penjaga ini terlebih dahulu. Mari kita lihat manfaatnya dari berbagai sisi:

- **Aspek Keamanan (Menjaga Toko dari Orang Jahat):**

- Dengan adanya penjaga di depan, orang dari luar tidak akan tahu siapa sebenarnya kasir kita, seperti apa ruang belakang (infrastruktur internal) kita, atau di mana brankas (database) kita disimpan. Reverse proxy menyembunyikan alamat IP asli dari server aplikasi kita, membuatnya jauh lebih sulit bagi peretas untuk menargetkan serangan secara langsung.
  - Penjaga bisa menyaring pengunjung yang mencurigakan sebelum mereka sempat masuk. Reverse proxy dapat berfungsi sebagai *Web Application Firewall* (WAF), yang mampu mendeteksi dan memblokir serangan umum seperti injeksi SQL atau *Cross-Site Scripting* (XSS) sebelum mereka mencapai aplikasi utama kita yang menyimpan data sensitif pasien.
  - Penjaga ini bisa mengurus semua urusan "gembok dan kunci" (enkripsi SSL/TLS). Ini memastikan semua komunikasi aman dan terenkripsi, dan kita bisa mengelolanya di satu tempat, bukan di setiap "kasir" secara terpisah.
- **Aspek Skalabilitas (Menambah Kasir Saat Ramai):** Bayangkan jika toko kita tiba-tiba sangat ramai. Tanpa penjaga, semua pengunjung akan menumpuk di satu kasir, menyebabkan antrean panjang dan kelambatan. Penjaga keamanan kita bisa dengan cerdas mengarahkan pengunjung ke kasir-kasir lain yang lebih kosong. Reverse proxy berfungsi sebagai *load balancer*, yang mendistribusikan lalu lintas ke beberapa server aplikasi. Jika satu server sibuk atau mati, lalu lintas akan dialihkan ke server lain yang sehat. Ini memastikan aplikasi SATUSEHAT tetap responsif dan tidak down bahkan saat diakses oleh banyak pengguna secara bersamaan, misalnya saat ada program vaksinasi massal.
- **Aspek Performa (Mempercepat Pelayanan):** Jika ada pengunjung yang sering menanyakan hal yang sama (misalnya, "di mana letak toilet?"), penjaga kita bisa langsung menjawabnya tanpa perlu bertanya ke kasir setiap saat. Reverse proxy dapat menyimpan konten yang sering diakses (seperti gambar, artikel, atau data statis) dalam *cache*. Ketika ada permintaan untuk konten tersebut, reverse proxy bisa langsung memberikannya tanpa perlu membebani server aplikasi. Ini membuat aplikasi terasa jauh lebih cepat bagi pengguna.
- **Aspek Kemudahan Pengelolaan (Menata Ulang Toko Tanpa Mengganggu Pengunjung):** Dengan adanya penjaga di depan, kita bisa dengan bebas

menata ulang atau merenovasi bagian dalam toko (misalnya, mengganti server, memperbarui teknologi) tanpa harus menutup toko. Pengunjung tetap berinteraksi dengan penjaga yang sama. Reverse proxy memberikan satu titik masuk yang konsisten. Tim developer bisa melakukan pemeliharaan, pembaruan, atau bahkan mengganti seluruh backend tanpa mengganggu layanan yang sedang berjalan. Ini memungkinkan kita untuk terus berinovasi dan meningkatkan SATUSEHAT dengan lebih lincah.

Kesimpulan untuk Stakeholder:

Dengan menambahkan reverse proxy, kita tidak hanya membangun benteng pertahanan yang lebih kuat untuk data sensitif pasien kita, tetapi juga menciptakan fondasi yang memungkinkan SATUSEHAT untuk tumbuh besar, melayani lebih banyak orang dengan cepat, dan beradaptasi dengan kebutuhan di masa depan tanpa gangguan. Ini adalah investasi strategis untuk keamanan, keandalan, dan skalabilitas bisnis kita.

6. (2.5 poin) Jelaskan mekanisme bagaimana SOCKS5 dapat establish TCP connections ke target servers melalui proxy layer. Mengapa SOCKS5 memerlukan *additional protocol layer* di atas TCP, dan apa *fundamental advantages* dari *proxy approach* dibandingkan *direct TCP connections*?

Ada serangkaian negosiasi yang terjadi antara klien dan proxy server, seperti yang didefinisikan dalam RFC 1928

#### 1. Koneksi Awal dan Negosiasi Autentikasi:

- Klien terlebih dahulu membuka koneksi TCP ke proxy server SOCKS5 (biasanya di port 1080).
- Setelah koneksi terjalin, klien mengirimkan "pesan sapaan" (*greeting message*). Pesan ini berisi versi protokol SOCKS (yaitu, `0x05` untuk SOCKS5) dan daftar metode autentikasi yang didukung oleh klien (misalnya, 'tanpa autentikasi', 'username/password', 'GSSAPI').
- Proxy server menerima pesan ini, memilih salah satu metode autentikasi yang juga didukungnya, dan mengirimkan kembali

respons yang berisi metode yang dipilih. Jika tidak ada metode yang cocok, server akan menolak koneksi

2. **Proses Autentikasi (Jika Diperlukan):** Jika metode yang dipilih bukan 'tanpa autentikasi', maka akan terjadi sub-negosiasi yang spesifik untuk metode tersebut. Misalnya, jika 'username/password' dipilih, klien akan mengirimkan kredensialnya, dan server akan memverifikasinya sebelum melanjutkan.
3. **Permintaan Koneksi dari Klien:** Setelah autentikasi berhasil (atau jika tidak diperlukan), klien mengirimkan permintaan koneksi (*connection request*). Pesan ini berisi informasi penting:
  - **Versi SOCKS:** Sekali lagi, 0x05.
  - **Perintah (Command):** 0x01 untuk 'establish a TCP/IP stream connection' (perintah yang paling umum digunakan).
  - **Alamat Tujuan:** Alamat dari server target yang ingin dihubungi. SOCKS5 sangat fleksibel; alamat ini bisa berupa alamat IPv4, IPv6, atau bahkan nama domain (yang akan di-resolve oleh proxy server, bukan oleh klien).
  - **Port Tujuan:** Nomor port di server target yang ingin dihubungi.
4. **Aksi oleh Proxy Server dan Respons:**
  - Setelah menerima permintaan, proxy server SOCKS5 akan mencoba membuka koneksi TCP ke alamat dan port tujuan yang diminta oleh klien.
  - Proxy server kemudian mengirimkan pesan balasan (*reply message*) kepada klien untuk memberitahukan status permintaan tersebut:
    - Jika koneksi ke server tujuan berhasil dibuat, balasannya akan berisi kode sukses (0x00) serta alamat IP dan port dari sisi proxy yang terikat ke koneksi tersebut.
    - Jika gagal (misalnya, host tidak dapat dijangkau, koneksi ditolak), balasannya akan berisi kode error yang sesuai.<sup>43</sup>
5. **Penyaluran Data (Relaying):** Jika koneksi berhasil, proxy server SOCKS5 sekarang bertindak sebagai penyalur transparan. Semua data yang dikirim

klien ke proxy akan diteruskan ke server tujuan, dan sebaliknya, semua data dari server tujuan akan diteruskan kembali ke klien. Dari sudut pandang klien dan server tujuan, mereka seolah-olah terhubung secara langsung

SOCKS5 memerlukan lapisan protokol tambahan di atas TCP karena ia beroperasi pada Session Layer (Layer 5) dari model OSI. Lapisan tambahan ini diperlukan untuk menyediakan fungsionalitas yang tidak dimiliki oleh TCP itu sendiri:

1. TCP hanya membangun koneksi end-to-end. Protokol SOCKS5 menambahkan langkah-langkah negosiasi di awal (seperti pemilihan metode autentikasi dan spesifikasi tujuan) yang memungkinkan proxying yang cerdas dan aman.
2. TCP tidak memiliki konsep autentikasi pengguna. SOCKS5 menambahkan lapisan ini untuk memastikan bahwa hanya pengguna yang berwenang yang dapat menggunakan layanan proxy.
3. TCP bekerja dengan alamat IP. SOCKS5 menambahkan kemampuan untuk meminta koneksi menggunakan nama domain, yang kemudian akan di-resolve oleh proxy. Ini sangat berguna karena memindahkan beban resolusi DNS ke proxy, yang bisa jadi lebih aman atau memiliki akses ke jaringan internal.

Pendekatan proxy seperti SOCKS5 menawarkan beberapa keunggulan fundamental dibandingkan koneksi TCP langsung:

1. **Melintasi Batasan Jaringan (Firewall Traversal):** Ini adalah salah satu kegunaan utamanya. Dalam banyak jaringan perusahaan, koneksi keluar dibatasi oleh *firewall*. Dengan mengarahkan semua lalu lintas melalui proxy server SOCKS5 yang diizinkan, aplikasi di dalam jaringan dapat mengakses layanan di internet luar seolah-olah tidak ada *firewall*.  
Proxy bertindak sebagai gerbang yang aman.
2. **Anonimitas dan Penyembunyian IP:** Ketika klien terhubung ke server tujuan melalui proxy, server tujuan hanya melihat alamat IP dari proxy server, bukan alamat IP asli klien. Ini memberikan lapisan anonimitas dan membantu melindungi identitas klien.
3. **Akses ke Jaringan Internal:** SOCKS5 dapat digunakan secara terbalik (sebagai reverse proxy atau melalui SSH tunneling) untuk memungkinkan pengguna dari luar mengakses layanan yang hanya tersedia di dalam

jaringan internal (misalnya, database atau server pengembangan) tanpa harus mengekspos layanan tersebut langsung ke internet.

4. **Fleksibilitas Protokol:** SOCKS5 bersifat agnostik terhadap protokol aplikasi. Karena beroperasi di Layer 5, ia dapat meneruskan hampir semua jenis lalu lintas TCP (dan juga UDP dengan perintah ASSOCIATE) tanpa perlu memahami konten datanya. Ini membuatnya lebih serbaguna daripada proxy tingkat aplikasi (seperti HTTP proxy) yang hanya memahami protokol spesifik.

**Referensi:** <https://en.wikipedia.org/wiki/SOCKS>

7. (4 poin) Misal kamu adalah seorang arsitek jaringan yang ditugaskan untuk merancang sistem routing internal (Interior Gateway Protocol/IGP) untuk dua klien yang sangat berbeda:
  - Klien A (Toko Ritel): Sebuah jaringan kecil dengan 5 router yang menghubungkan kasir, gudang, dan kantor manajer.
  - Klien B (Pusat Data): Sebuah jaringan perusahaan besar yang kompleks dengan 50 router, jalur redundan, koneksi fiber optik, dan membutuhkan kecepatan pemulihan yang kuat saat ada link yang putus.

Kamu memiliki dua pilihan jenis protokol routing utama: **Distance Vector** dan **Link State**. Dari dua jenis pilihan tersebut:

- a. (1.5 poin) Untuk jaringan Klien A, filosofi protokol mana yang lebih cocok? Mengapa cara kerja fundamental protokol tersebut lebih sesuai dibanding protokol satunya lagi?

**Distance Vector** lebih cocok dan praktis. Setiap router saling mengomunikasikan jaraknya kepada router lain melalui tetangganya (Nanti tetangganya masing-masing bangun tabelnya sendiri). Ini gampang di setup di lingkungan dengan perangkat yang sedikit.

Lalu, kelemahan utama *Distance Vector*, seperti konvergensi yang lambat dan masalah "Count to Infinity", tidak terlalu menjadi masalah di jaringan sekecil ini. Dengan hanya 5 router, kabar perubahan topologi (misalnya, satu link putus) akan menyebar dengan cukup cepat, dan risiko *routing loop* yang persisten jauh lebih kecil.

- b. (1.5 poin) Untuk jaringan Klien B, mana yang menjadi pilihan yang jelas? Bagaimana mekanisme protokol ini secara langsung menjawab kebutuhan?

Jaringan pusat data yang kompleks dengan 50 router, artinya jelas perlu jalur redundan, dan kebutuhan pemulihan cepat. Ini cocok buat mekanisme Link State.

- Setiap router dalam jaringan *Link State* membangun dan memelihara **peta topologi lengkap** dari seluruh areanya. Ini dicapai dengan setiap router mengirimkan *Link State Advertisements* (LSA) yang berisi informasi tentang dirinya dan koneksi langsungnya. LSA ini kemudian di-flood ke seluruh jaringan, sehingga setiap router memiliki pandangan yang identik dan akurat tentang jaringan.
- Ketika sebuah *link* putus, *router* yang terhubung ke *link* tersebut akan segera mengirimkan LSA baru untuk mengumumkan perubahan tersebut. LSA ini akan menyebar dengan cepat ke seluruh jaringan.

- c. (1 poin) Salah satu kelemahan fatal dari protokol Distance Vector adalah masalah "Count to Infinity". Mengapa protokol Link State, berdasarkan desain dasarnya, secara inheren kebal terhadap masalah spesifik ini?

*Link State* tidak bergantung pada kabar dari tetangga. Setiap router memiliki **peta topologi yang lengkap dan independen**.

- Informasi tentang sebuah *link* (misalnya, *link* antara *router C* dan *D* putus) hanya berasal dari *router* yang terhubung langsung ke *link* tersebut (yaitu, *C* dan *D*). *Router* lain tidak akan membuat atau menyebarluaskan informasi tentang *link* yang tidak terhubung langsung dengannya.
- Ketika *link C-D* putus, *C* dan *D* akan mengirimkan LSA baru yang menyatakan bahwa *link* tersebut tidak lagi aktif. LSA ini di-flood ke seluruh jaringan. Setiap *router* (*A, B, E, dst.*) menerima LSA ini dan memperbarui petanya sendiri. Kemudian, setiap *router* secara mandiri menjalankan algoritma SPF pada peta yang sudah diperbarui tersebut untuk menemukan jalur terbaik yang baru.

- Karena setiap router menghitung jalurnya berdasarkan peta yang sama dan akurat, tidak ada kemungkinan bagi router B untuk berpikir ia bisa mencapai C melalui A jika jalur A ke C juga sudah tidak ada di peta. Loop seperti pada Distance Vector tidak dapat terbentuk karena keputusan routing didasarkan pada pengetahuan topologi yang lengkap, bukan pada metrik jarak yang diiklankan oleh tetangga

8. (3 poin) Anda sudah belajar salah satu cara mem-bypass content filter Kominfo, yaitu via DNS over HTTPS (DoH). Namun, sekitar 2023, Kominfo mengeluarkan sebuah jurus pamungkas, sesuatu yang lebih dahsyat daripada sekadar *blocking IP* atau *DNS poisoning*, yaitu [Deep Packet Inspection + TCP Reset Attack](#).

Untungnya, para pengguna *internet* di dunia sudah jauh lebih modern daripada manusia-manusia di Kominfo, jadi jurus pamungkas itu sudah tidak level bagi kita yang sudah punya jurus untuk meng-counter-nya.

Kita bisa menggunakan VPN untuk mengatasi serangan kominfo. Namun, TCP Reset Attack itu bekerja dengan mengirim *TCP reset packet* palsu ke *client* dan *server*. Karena VPN bekerja sebagai *proxy*, *client* di sini tetap sama, dan *server* berbeda (*server* sekarang adalah *server VPN*, bukan *server situs tujuan*). Lantas, apa yang menyebabkan serangan kominfo berhasil dilakukan ketika tujuannya adalah *server situs terlarang*, tetapi gagal untuk *server VPN*?

Serangan ini berhasil untuk situs terlarang karena sistem DPI dapat melihat tujuan akhir dari komunikasi Anda. Ketika Anda terhubung langsung ke [situs-terlarang.com](#) (misal [reddit.com](#)), DPI dapat membaca nama domain ini di dalam paket data (misalnya, dalam permintaan DNS, TLS Client Hello, atau HTTP Host header). Setelah tujuan teridentifikasi sebagai target blokir, sistem DPI tahu persis ke mana harus mengirim paket RST palsu: ke alamat IP Anda dan alamat IP [situs-terlarang.com](#).

Ketika menggunakan VPN, tercipta sebuah terowongan terenkripsi (encrypted tunnel) antara komputer dan server VPN. Semua lalu lintas internet komputer tersebut akan dimasukkan ke dalam terowongan ini.

- Seluruh paket data, termasuk informasi tentang tujuan akhirnya (misalah, permintaan ke [situs-terlarang.com](#)), dienkripsi secara menyeluruh sebelum meninggalkan komputer.

- Ketika paket terenkripsi ini melewati sistem DPI milik ISP, yang bisa dilihat oleh DPI hanyalah:
  1. **Sumber:** Alamat IP Anda.
  2. **Tujuan:** Alamat IP dari **server VPN**.
  3. Isi Paket: Data acak yang terenkripsi.  
DPI tidak bisa melihat tujuan sebenarnya ([situs-terlarang.com](#)) karena informasi tersebut tersembunyi di dalam "terowongan" yang terenkripsi.
- Karena DPI tidak tahu tujuan akhir Anda, ia tidak bisa mengirimkan paket RST ke [situs-terlarang.com](#). Ia hanya bisa mencoba mengirim paket RST untuk memutus koneksi antara Anda dan server VPN. Namun, banyak protokol VPN modern (seperti OpenVPN atau WireGuard) berjalan di atas UDP, bukan TCP, yang membuat serangan TCP Reset sama sekali tidak relevan.

9. (2 poin) Suatu hari kamu mencoba melakukan ping ke [hiyotoromoe.com](#) dan mendapatkan balasan yang normal, dengan waktu respons yang normal. Namun, ketika kamu mencoba curl [hiyotoromoe.com](#), perintah tersebut gagal.

a. (0.5 poin) Jelaskan protokol dan lapisan jaringan yang digunakan oleh ping dan curl!

- ping: memakai ICMP dan berada pada lapisan jaringan ketiga (Network Layer)
- curl: memakai HTTPS/HTTP dan berada pada lapisan jaringan keempat (Transport Layer)

b. (0.5 poin) Mengapa skenario seperti di atas bisa terjadi? Sebutkan dua kemungkinan penyebab, dan bagaimana cara kamu bisa menyelidikinya.

1. **Firewall Memblokir Port TCP:** *firewall* Jaringan telah dikonfigurasi untuk mengizinkan lalu lintas ICMP (yang digunakan ping) untuk keperluan diagnostik, tetapi secara eksplisit memblokir port TCP yang digunakan oleh layanan web, yaitu **port 80 (untuk HTTP)** dan **port 443 (untuk HTTPS)**. Jadi, ICMP Echo Request bisa lewat, tetapi

upaya koneksi TCP dari `curl` ke port 80 atau 443 akan ditolak atau diabaikan oleh *firewall*

Untuk menyelidikinya, bisa pakai `telnet` untuk memeriksa apakah port tujuan terbuka. Perintah seperti `telnet hiyotoromoe.com 80` akan menunjukkan apakah koneksi ke port tersebut bisa dibuat. Jika gagal, kemungkinan besar ada *firewall* yang memblokir.

2. **Server Web Tidak Berjalan atau Rusak:** `ping` hanya menguji apakah host (mesin server) itu sendiri aktif dan dapat dijangkau di jaringan. Ia tidak tahu apa-apa tentang layanan atau aplikasi yang berjalan di atasnya. Bisa jadi mesin servernya hidup dan sehat (sehingga merespons `ping`), tetapi layanan web server (seperti Apache atau Nginx) sedang mati, crash, atau salah konfigurasi sehingga tidak "mendengarkan" di port 80/443. Akibatnya, `curl` akan gagal terhubung karena tidak ada aplikasi yang merespons di port tujuan.

Jika punya akses ke server, periksa status layanan web server (misalnya, `systemctl status nginx`). Periksa juga log kesalahan (error log) dari web server.

- c. (0.5 poin) Konfigurasi jaringan atau *firewall* seperti apa yang bisa menyebabkan gejala ini terjadi?

Konfigurasi yang paling mungkin adalah aturan *firewall* (baik di *hardware firewall* jaringan atau *software firewall* di host itu sendiri, seperti `iptables`) sebagai berikut:

- **ALLOW** `INPUT protocol=icmp icmp-type=echo-request`
- **DROP** atau **REJECT** `INPUT protocol=tcp dport=80`
- **DROP** atau **REJECT** `INPUT protocol=tcp dport=443`

- d. (0.5 poin) Pada hari lain, saat kamu mengatur host-only network di VirtualBox pada mesin host Windows, kamu menyadari bahwa VM tidak bisa melakukan ping ke host, tetapi `curl` ke server web lokal di host berhasil. Ceritakan kemungkinan penyebab hal ini terjadi!

Hampir pasti disebabkan oleh **konfigurasi firewall di mesin host**. Salah satunya, karena *Firewall* di sistem operasi *host* (misalnya, Windows Defender Firewall di Windows) sering kali memiliki aturan *default* untuk memblokir **lalu lintas ICMP Echo Request yang masuk (*inbound*)** sebagai tindakan keamanan dasar untuk mengurangi "visibilitas" mesin di jaringan dan mencegah serangan *smurf attack*.

## II. Sistem Paralel dan Terdistribusi



Distributed and Parallel [Sisters](#)

1. (2.5 poin) Dibanding dengan mesinmu sendiri, terkadang membuka file yang disimpan pada *distributed file system* memakan waktu yang jauh lebih lama.
  - a. Sebutkan dua area di mana *bottleneck* ini mungkin terjadi!
  - b. Bagaimana masalah ini dapat diatasi? (hint: bisa me-refer ke *filesystem* yang sudah ada, seperti GFS atau HDFS)
2. (2.5 poin) Jelaskan bagaimana vector clock dapat mendeteksi concurrent events dan causal relationships. Mengapa vector clock lebih *powerful* daripada *logical clock* untuk *causal ordering*?

Cara kerja Vector clock adalah sebagai berikut:

- **Inisialisasi:** Semua *clock* di semua vektor dimulai dari nol.
- **Kejadian Internal:** Setiap kali sebuah proses melakukan kejadian internal, ia menaikkan nilai *clock*-nya sendiri di dalam vektornya.
- **Pengiriman Pesan:** Sebelum mengirim pesan, sebuah proses menaikkan *clock*-nya sendiri, lalu melampirkan seluruh vektor *clock*-nya ke pesan tersebut.

- **Penerimaan Pesan:** Ketika sebuah proses menerima pesan, ia pertama-tama menaikkan *clock*-nya sendiri. Kemudian, untuk setiap entri di vektornya, ia memperbarui nilainya dengan mengambil nilai **maksimum** antara entri di vektornya sendiri dan entri yang sesuai di vektor yang diterima bersama pesan.

Dengan mekanisme ini, kita bisa menentukan hubungan antar kejadian:

- **Hubungan Kausal (Causal Relationship):** Sebuah kejadian A dikatakan **menyebabkan** kejadian B (ditulis  $A \rightarrow B$ ) jika vektor *clock* A lebih kecil dari vektor *clock* B. Ini berarti setiap elemen di vektor A kurang dari atau sama dengan elemen yang sesuai di vektor B, dan setidaknya ada satu elemen yang benar-benar lebih kecil. Ini secara akurat menangkap alur "terjadi sebelum" (*happened-before*).
- **Kejadian Konkuren (Concurrent Events):** Dua kejadian A dan B dikatakan **konkuren** (terjadi secara bersamaan atau tanpa hubungan sebab-akibat) jika vektor *clock* mereka tidak dapat dibandingkan. Artinya, baik  $A \rightarrow B$  maupun  $B \rightarrow A$  tidak benar. Ini terjadi ketika beberapa elemen di vektor A lebih besar dari B, sementara beberapa elemen lain di vektor B lebih besar dari A.

Perbedaan Vector *clock* dengan Logical Clock:

- **Lamport Clock:** Jika  $A \rightarrow B$ , maka  $C(A) < C(B)$ . Namun, jika  $C(A) < C(B)$ , kita **tidak bisa** menyimpulkan bahwa  $A \rightarrow B$ . A dan B bisa saja konkuren.
- **Vector Clock:** Hubungannya adalah "jika dan hanya jika".  $A \rightarrow B$   **jika dan hanya jika**   $V(A) < V(B)$ .

Dengan kata lain, vector *clock* secara sempurna menangkap kausalitas. Ia tidak hanya dapat memastikan hubungan sebab-akibat tetapi juga, yang lebih penting, **dapat secara definitif mendeteksi ketika dua kejadian tidak memiliki hubungan sebab-akibat (yaitu, konkuren)**.

3. (2.5 poin) Sistem terdistribusi tidak memiliki satu sumber waktu global yang disepakati oleh semua node. Menurut pendapatmu, apa konsekuensi dari tidak adanya *global clock* ini terhadap sinkronisasi proses, event ordering, dan koordinasi antar node?

### **1. Pengurutan Kejadian (Event Ordering):**

- Tanpa jam global, mustahil untuk menentukan urutan absolut dari dua kejadian yang terjadi di dua mesin yang berbeda. Jika node A mencatat kejadian pada pukul 10:00:00.050 dan node B mencatat kejadian lain pada pukul 10:00:00.040, kita tidak bisa langsung menyimpulkan bahwa kejadian di B terjadi lebih dulu. Bisa jadi, *clock* di A sedikit lebih cepat daripada *clock* di B.
- Akibatnya, sistem terdistribusi harus beralih dari konsep "urutan total" (di mana setiap kejadian memiliki posisi pasti di garis waktu) ke "urutan parsial" atau "urutan kausal". Yang bisa kita pastikan hanyalah urutan kejadian yang memiliki hubungan sebab-akibat (misalnya, kejadian "pesan diterima" pasti terjadi setelah kejadian "pesan dikirim"). Kejadian yang tidak saling berhubungan dianggap konkuren.

### **2. Konsistensi Data (Data Consistency):**

- Banyak model konsistensi data, terutama model yang kuat seperti *linearizability* (konsistensi kuat), secara implisit mengasumsikan adanya garis waktu global. Tanpa itu, jika dua klien secara bersamaan menulis nilai yang berbeda ke item data yang sama yang direplikasi di beberapa *node*, akan sulit untuk menentukan tulisan mana yang "terakhir" dan harus menjadi nilai final.
- Dalam sistem seperti *database* terdistribusi atau sistem keuangan, urutan transaksi sangat penting. Jika *clock* tidak sinkron, transaksi bisa jadi diterapkan dalam urutan yang salah di *node* yang berbeda, yang dapat merusak integritas data dan menyebabkan kerugian finansial.

### **3. Koordinasi Antar Node (Coordination):**

- Banyak algoritma terdistribusi bergantung pada waktu, misalnya, untuk mekanisme *timeout*. Jika sebuah *leader* dalam algoritma konsensus gagal mengirimkan *heartbeat*, *node* lain akan menunggunya selama periode *timeout* sebelum memulai pemilihan *leader* baru. Jika *clock* di *node-node* tersebut tidak sinkron, beberapa *node* mungkin menganggap *leader* sudah mati terlalu

- cepat, sementara yang lain masih menunggunya, yang dapat menyebabkan kekacauan dan kondisi *split-brain*.
- Mengimplementasikan kunci terdistribusi (*distributed lock*) untuk memastikan hanya satu proses yang dapat mengakses sumber daya kritis pada satu waktu menjadi lebih kompleks. Algoritma yang berbasis *timestamp* untuk memberikan giliran akan gagal jika *timestamp* tidak dapat diandalkan.

4. (3 poin) Apa itu False Sharing? Mengapa ia disebut 'silent killer' dalam parallel programming?

**Follow up question:** "Jika dua thread pada CPU berbeda mengakses variabel A dan B yang terletak dalam cache line yang sama, mengapa performa turun drastis meskipun tidak ada race condition? Jelaskan solusinya!"

False sharing adalah sebuah fenomena ketika ada dua atau lebih thread yang berjalan di inti CPU yang berbeda, secara independen memodifikasi variabel yang berbeda, tetapi variabel-variabel tersebut kebetulan berada di dalam baris cache (cache line) yang sama. Ia disebut "pembunuh senyap" karena **Susah dideteksi + gabakal Race Condition**. Dari logika, kodennya benar. Setiap *thread* bekerja pada datanya sendiri, jadi gaada race condition atau masalah sinkronisasi yang jelas. Kode tetep bakal hasilin output yang benar, tetapi berjalan jauh lebih lambat dari yang diharapkan.

Jawaban follow up:

- **Cara Kerja Cache CPU:** CPU tidak membaca memori (RAM) per byte. Ia membaca dan menulis data dalam blok-blok berukuran tetap yang disebut **cache line** (biasanya 64 byte). Ketika *thread* 1 di CPU 1 ingin mengakses variabel A, seluruh *cache line* yang berisi A (dan juga B) akan disalin dari RAM ke *cache L1* milik CPU 1.
- **Protokol Koherensi Cache (Contoh: MESI):** Dalam sistem *multi-core*, setiap inti CPU memiliki cache-nya sendiri. Untuk menjaga konsistensi data di semua cache, ada protokol yang disebut protokol koherensi cache (misalnya, MESI). Protokol ini memastikan bahwa jika satu inti CPU mengubah data, salinan data di cache inti lain menjadi tidak valid.
- **Skenario False Sharing:**

- **Langkah 1:** Thread 1 di CPU 1 menulis ke variabel A. Cache *line* yang berisi A dan B dimuat ke cache CPU 1 dan ditandai sebagai "Modified".
  - **Langkah 2:** Thread 2 di CPU 2 ingin menulis ke variabel B. Karena B berada di cache *line* yang sama dengan A, CPU 2 harus mendapatkan kepemilikan eksklusif atas cache *line* tersebut.
  - **Langkah 3 (Ping-Pong Effect):** Protokol koherensi cache akan memaksa CPU 1 untuk menulis kembali (*flush*) seluruh cache *line* tersebut ke memori utama (atau cache L3), dan kemudian membatalkan (*invalidate*) salinannya di cache CPU 1. Setelah itu, CPU 2 akan memuat cache *line* tersebut dari memori dan menandainya sebagai "Modified" di cache-nya.
  - **Langkah 4:** Sekarang, jika thread 1 ingin menulis ke A lagi, proses yang sama akan terulang kembali ke arah sebaliknya. Cache *line* akan di-*flush* dari CPU 2, di-*invalidate*, dan dimuat kembali oleh CPU 1.
- Proses "ping-pong" di mana seluruh cache *line* (64 byte) harus bolak-balik berpindah antar inti CPU melalui bus memori—hanya karena thread yang berbeda ingin memodifikasi byte yang berbeda di dalamnya—inilah yang menyebabkan penurunan performa yang drastis. Setiap operasi tulis yang seharusnya sangat cepat (hanya ke cache L1) menjadi operasi yang sangat lambat karena melibatkan komunikasi antar-inti dan akses ke level cache yang lebih tinggi atau bahkan RAM.
- Solusinya adalah memastikan bahwa variabel yang akan dimodifikasi secara independen oleh thread yang berbeda tidak berada di cache *line* yang sama. Ini bisa dicapai dengan:
  - **Padding:** Menambahkan byte kosong (yang tidak digunakan) di antara variabel A dan B sehingga mereka dipaksa berada di cache *line* yang berbeda. Misalnya, jika A adalah `int` (4 byte) dan cache *line* berukuran 64 byte, kita bisa menambahkan `char padding` setelah A untuk memastikan B akan berada di cache *line* berikutnya
  - **Alignment:** Menggunakan direktif kompilator atau fitur bahasa (seperti `alignas` di C++) untuk secara eksplisit memberitahu

komplilator agar menempatkan sebuah variabel di awal dari sebuah cache line baru.

5. (3 poin) [Video](#). Setelah menonton video tersebut, kita asumsikan komunikasi antar layanan utama seperti **Image**, **Metadata**, **Feed**, **Like**, **Comment**, dan **Fanout** dilakukan melalui **Remote Procedure Call (RPC)**, sementara proses **Fanout** ke followers dilakukan secara **asinkron** melalui **Message Queue** (kalau ini sesuai dengan video).

Bagaimana layanan-layanan seperti **Image**, **Metadata**, dan **Feed** saling berinteraksi menggunakan RPC? Dan mengapa **Fanout** ke followers sebaiknya diproses secara asinkron menggunakan Message Queue, bukan RPC langsung?

Interaksi RPC bersifat sinkron dan request-response.

- Ketika seorang pengguna membuka aplikasi Instagram, layanan **Feed** perlu membangun *timeline* untuk pengguna tersebut. Untuk melakukan ini, ia akan:
  - Membuat panggilan RPC ke layanan **Metadata** untuk mendapatkan daftar ID postingan dari orang-orang yang di-follow oleh pengguna. Panggilan ini mungkin terlihat seperti `get_post_ids_for_user(user_id)`.
  - Layanan **Feed** akan menunggu respons dari layanan **Metadata**.
  - Setelah mendapatkan daftar ID, layanan **Feed** akan membuat panggilan RPC lagi, mungkin ke layanan **Image** dan layanan **Metadata** (atau layanan Post), untuk mengambil detail dari setiap postingan (URL gambar, caption, jumlah likes, dll) untuk setiap ID yang didapat.
  - Layanan **Feed** mengumpulkan semua respons ini, menyusunnya menjadi feed, dan mengirimkannya kembali ke aplikasi klien.
- Interaksi ini menggunakan RPC karena bersifat **transaksional** dan **membutuhkan respons segera**. Layanan **Feed** tidak bisa melanjutkan tugasnya (membangun *timeline*) sebelum ia mendapatkan data dari layanan lain. Komunikasi ini bersifat erat (*tightly coupled*) dalam hal waktu; pemanggil menunggu pemanggil.

Proses Fanout sangat berbeda dan sebaiknya dilakukan secara asinkron menggunakan Message Queue. **Alasan Asinkron:**

- Seorang pengguna bisa memiliki jutaan pengikut. Jika layanan *upload* harus membuat jutaan panggilan RPC secara langsung untuk menyisipkan postingan baru ke *timeline* setiap pengikut dan menunggu semuanya selesai, proses *upload* akan memakan waktu sangat lama dan kemungkinan besar akan gagal.
- Dengan Message Queue, prosesnya menjadi:
  - Layanan *upload* hanya melakukan satu tugas cepat: mempublikasikan sebuah pesan (misalnya, {"user\_id": "A", "post\_id": "123"}) ke sebuah *topic* di Message Queue. Setelah itu, tugasnya selesai, dan ia bisa langsung memberikan respons "Upload berhasil" kepada pengguna.
  - Di sisi lain, ada layanan **Fanout** (atau sekelompok *worker*) yang bertugas "mendengarkan" *topic* tersebut. Ketika pesan baru masuk, *worker* akan mengambilnya dan mulai melakukan pekerjaan berat: menyebarkan ID postingan ke *timeline* jutaan pengikut.
  - Pendekatan ini memisahkan (*decouple*) proses *upload* dari proses distribusi. Jika layanan **Fanout** sedang lambat atau bahkan mati sementara, proses *upload* tidak akan terpengaruh. Pesan akan tetap aman di dalam queue dan akan diproses ketika layanan **Fanout** pulih. Ini membuat sistem secara keseluruhan jauh lebih tangguh.
- **Manajemen Beban (Load Management):** Message Queue bertindak sebagai *buffer*. Jika tiba-tiba ada banyak sekali postingan baru (misalnya, saat konser besar), queue akan menampung lonjakan pesan tersebut, dan para *worker* **Fanout** dapat memprosesnya sesuai dengan kapasitas mereka, mencegah sistem dari kelebihan beban.

Singkatnya, RPC digunakan untuk interaksi yang membutuhkan **jawaban langsung dan bersifat transaksional**, sementara Message Queue digunakan untuk pekerjaan yang bisa **ditunda, berjalan di latar belakang, dan perlu didistribusikan ke banyak tujuan** secara andal dan skalabel.

6. (3 poin) [Artikel](#). Setelah membaca artikel tersebut, jelaskan konklusi kamu (seolah-olah kamu sedang menceritakannya ke teman kamu yang kuliah di jurusan bisnis dan merupakan investor crypto akut) terkait perbedaan sistem terdistribusi dan terdesentralisasi.

Oke, bro, jadi lo kan suka banget sama dunia crypto. Nah, biar makin ngerti, ada dua istilah penting yang sering ketuker: **terdistribusi** dan **terdesentralisasi**.

Gampangnya gini:

- Sistem Terdistribusi itu Kayak Gojek.  
Bayangan Gojek. Driver-nya ada di mana-mana, tersebar di seluruh kota (ini bagian terdistribusi-nya). Mereka semua terhubung ke jaringan dan berbagi beban kerja. Kalau satu driver mogok, masih banyak driver lain yang bisa ambil orderan lo. Jadi, sistemnya tetap jalan dan efisien karena pekerjaannya disebar.  
Tapi, yang ngatur semuanya siapa? Tetap ada satu "otak" pusat, yaitu kantor pusat Gojek. Mereka yang punya aturan main, mereka yang nentuin tarif, mereka yang verifikasi driver, dan mereka yang bisa nge-ban akun lo. Jadi, meskipun komponennya tersebar, kontrolnya tetap terpusat. Google, Facebook, server bank—semuanya itu sistem terdistribusi.
- Sistem Terdesentralisasi itu Kayak Bitcoin.  
Nah, ini dunianya crypto. Di jaringan Bitcoin, server (yang disebut node) juga tersebar di seluruh dunia (jadi, dia juga terdistribusi). Tapi, beda kuncinya di sini: gaada satu pun "kantor pusat". Gaada satu perusahaan atau satu orang pun yang bisa mengubah aturan main seenaknya. Semua keputusan penting dibuat berdasarkan konsensus atau kesepakatan mayoritas dari semua peserta jaringan. Karena gaada pusatnya, gaada yang bisa "dimatikan". Pemerintah nggak bisa datengin satu kantor dan bilang, "Tutup Bitcoin!" karena "kantornya" ada di mana-mana dan ga dimiliki siapa-siapa.

**Jadi, Kesimpulannya Buat Lo sebagai Investor:**

- Semua sistem terdesentralisasi (kayak crypto) itu **pasti** terdistribusi (karena server-nya nyebar).
- Tapi, nggak semua sistem terdistribusi itu terdesentralisasi (kayak Gojek atau bank, yang masih punya bos).

### III. Teknologi Sistem Terintegrasi



Integrating the freeze team wheelchair

Penilaian pada soal berikut akan mengacu pada standar mata kuliah STI. Jawaban yang bertele-tele, terlalu panjang, dan menunjukkan kurangnya pemahaman konsep oleh penulis dapat mempengaruhi nilai secara negatif.

1. **(20 Poin)** Perusahaan bernama Furina Courthouse Corporated .inc adalah sebuah perusahaan yang bergerak di sektor layanan hukum digital dengan fokus utama pada efisiensi proses peradilan dan legalitas melalui transformasi digital. Visi perusahaan ini adalah menjadi pioneer dalam modernisasi pelayanan hukum di seluruh Fontaine dengan mengakomodasi kebutuhan masyarakat yang semakin terdigitalisasi.

**Model bisnis** FCC berupa B2B (Business to Business) dan B2G (Business to Government) dengan menyediakan platform dan layanan kepada:

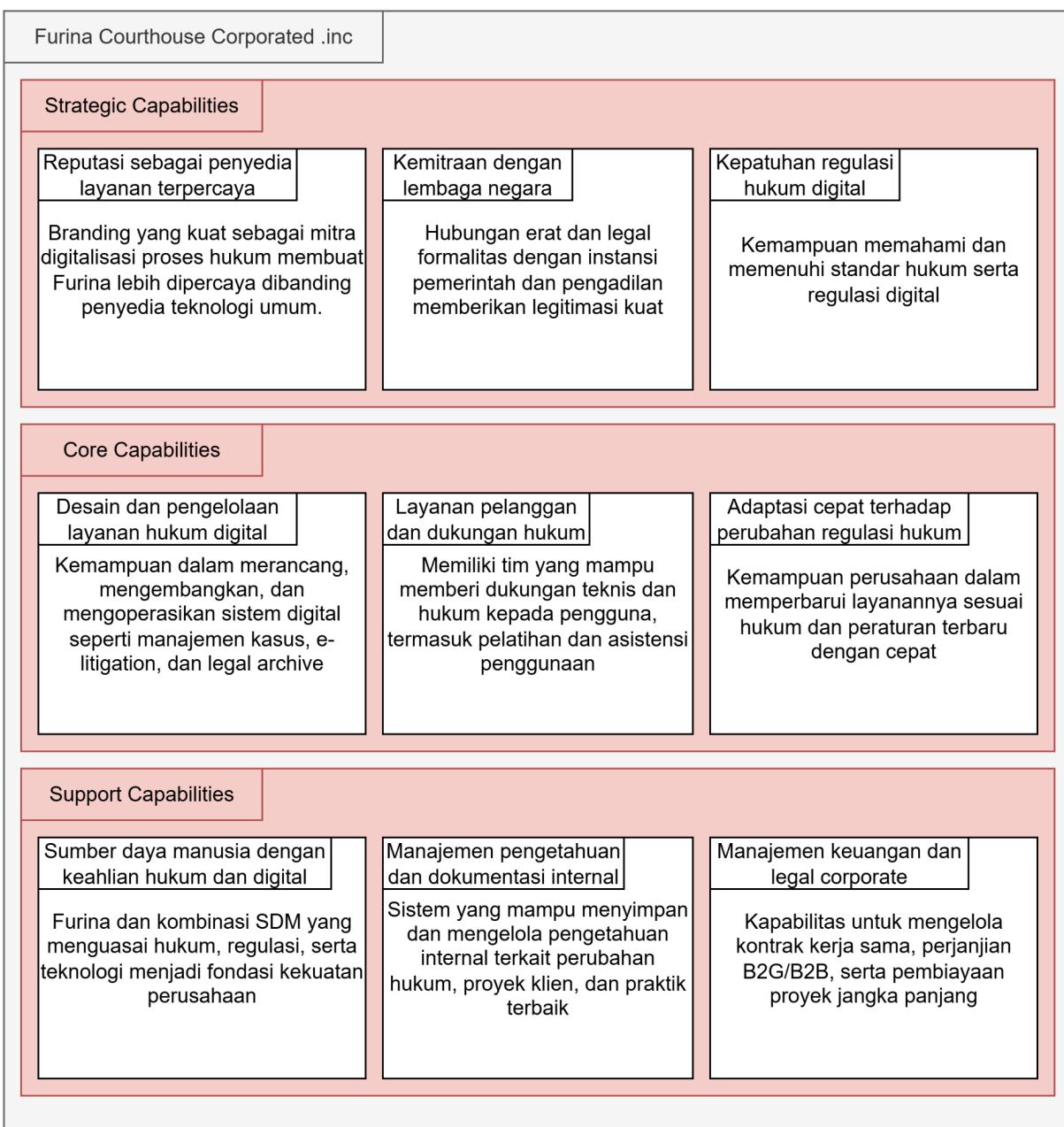
- Opera Epiclese (Mahkamah Agung Fontaine) dan Gardes (Kepolisian Fontaine) untuk digitalisasi proses hukum, administrasi, dan manajemen arsip hukum.
- Perusahaan besar, terutama yang memiliki kebutuhan legalisasi dan compliance yang tinggi.

**Layanan utama** yang diberikan oleh FCC antara lain:

- Layanan manajemen kasus untuk pelacakan dan dokumentasi proses hukum dari awal hingga keputusan.
- Digital legal archive untuk menyimpan dan manajemen arsip hukum.
- E-Litigation platform untuk memfasilitasi proses persidangan jarak jauh dengan bukti elektronik.
- Layanan otentikasi & validasi dokumen untuk legalisasi digital dokumen hukum dan tanda tangan elektronik.

**Strategi bisnis** yang dimiliki oleh FCC berupa kemitraan strategis dengan institusi negara untuk memperkuat legitimasi serta layanan konsultasi hukum secara digital untuk lembaga atau perusahaan yang baru mulai melakukan transformasi digital dalam operasional hukumnya.

**Business Capability Model** dari FCC adalah sebagai berikut:



- Bagaimana arsitektur sistem yang harus dibuat agar semua lembaga hukum (Opera Epiclese dan Gardes) dapat saling berjalan secara efisien, mempunyai skalabilitas yang tinggi, dan memiliki interoperabilitas yang tinggi. Buatlah dan jelaskan diagram dari arsitektur sistem tersebut!
- FCC berencana bekerja sama dengan smart city untuk mengintegrasikan data dari IoT (seperti kamera lalu lintas aquabus, bodycam gardes, dan sensor kejadian). Bagaimana arsitektur sistem terintegrasi harus dibangun agar data tersebut dapat digunakan secara sah dan real-time di pengadilan?

- c. Apa tantangan keamanan dan privasi ketika sistem hukum digital seperti yang dikembangkan pada FCC harus mengintegrasikan perangkat IoT tersebut?
  - d. Jelaskan bagaimana FCC dapat menjamin keabsahan bukti elektronik yang diambil dari perangkat IoT agar tidak dipengaruhi oleh faktor internal (misalnya oknum penegak hukum atau lembaga sendiri)?
  - e. Dengan meningkatnya volume data dari sistem terintegrasi dan IoT, bagaimana FCC sebaiknya mengatur strategi skalabilitas dan efisiensi biaya penyimpanan tanpa mengorbankan integritas dan ketersediaan data untuk keperluan hukum.
  - f. Evaluasi risiko dan manfaat dari mengintegrasikan sistem peradilan digital dengan perangkat IoT di lapangan. Apa langkah mitigasi yang harus disiapkan Furina sebagai penyedia platform?
  - g. Berdasarkan arsitektur sistem dan evaluasi risiko yang telah anda susun, identifikasi area-area yang masih bisa dikembangkan atau diperbaiki agar sistem terintegrasi FCC menjadi lebih aman dan mudah diterima oleh stakeholder. Buatlah penjelasan (non-teknis) yang dapat disampaikan kepada stakeholder non-teknis seperti hakim, jaksa, atau pejabat instansi pemerintah agar mereka dapat paham!
2. **(2 poin)** Jika kamu bekerja untuk suatu perusahaan kamu pasti akan menemukan sebuah legacy code. Legacy code adalah kode program yang sudah tua atau usang, tetapi masih digunakan dalam suatu sistem. Kenapa integrasi terhadap legacy sistem sering gagal walaupun secara teknis API telah diimplementasikan dan tinggal digunakan?

Kegagalan ini jarang disebabkan oleh masalah teknis murni, melainkan lebih sering berakar pada masalah non-teknis yang kompleks dan sering diremehkan. Berikut adalah beberapa alasan utamanya:

#### 1. **Kurangnya Dokumentasi dan Pengetahuan Institusional**

Dokumentasi arsitektur, logika bisnis, dan bahkan cara kerja API-nya legacy code mungkin sudah tidak ada, tidak lengkap, atau sudah usang. Lebih buruk lagi, para developer asli yang membangun dan memahami sistem tersebut mungkin sudah lama pensiun atau pindah perusahaan. Akibatnya, tim saat ini mencoba menebak-nebak bagaimana sistem bekerja, apa efek samping dari memanggil API tertentu. Tanpa pemahaman yang mendalam

ini, integrasi yang dibangun seringkali rapuh dan penuh dengan asumsi yang salah.

## 2. Resistensi terhadap Perubahan dari "Penjaga" Sistem

Setiap *legacy system* biasanya memiliki "penjaga" atau tim pemeliharaan yang telah merawatnya selama bertahun-tahun. Mereka seringkali memiliki pola pikir "jika tidak rusak, jangan diperbaiki". **Resistensi budaya dan organisasi** ini bisa menjadi penghalang yang lebih besar daripada tantangan teknis apa pun.

## 3. Ketidakcocokan Proses Bisnis dan Logika Tersembunyi

API mungkin secara teknis mengembalikan data, tetapi data tersebut mungkin tidak sesuai dengan proses bisnis modern. *Legacy system* seringkali dibangun di sekitar alur kerja dan aturan bisnis dari era yang berbeda. Selain itu, seringkali ada "logika tersembunyi" di dalam *legacy code*—aturan-aturan bisnis yang tidak terdokumentasi yang penting untuk operasi yang benar. API mungkin tidak mengekspos logika ini, sehingga ketika sistem baru berinteraksi dengannya, hasil yang tidak diharapkan atau bahkan korupsi data bisa terjadi.

## 4. Masalah Performa dan Skalabilitas yang Tidak Terduga

API mungkin berfungsi dengan baik saat diuji dengan satu atau dua permintaan. Namun, *legacy system* di belakangnya mungkin tidak pernah diperuntukkan untuk menangani beban kerja yang dihasilkan oleh aplikasi modern. Ketika sistem baru mulai mengirimkan ratusan atau ribuan permintaan per menit melalui API, *legacy system* bisa menjadi sangat lambat atau bahkan crash.

#### IV. Keamanan Informasi



She doesn't have anything to do with information security, I just really love her - Duke

Untuk setiap soal, berikan jawaban sesingkat dan sepadat mungkin. Jawaban yang terlalu bertele-tele berpotensi di-O-kan.

1. (4 poin) Meskipun sebenarnya berbeda, konsep-konsep keamanan informasi berikut seringkali dianggap sama dan atau saling tertukar. Menggunakan referensi, berikan definisi masing-masing konsep, beserta dengan bagaimana setiap konsep (dalam pasangan yang sama) dapat saling melengkapi atau berlawanan!

##### a. Security vs Privacy

- Security (Keamanan) merujuk pada perlindungan data dari akses yang tidak sah (unauthorized access).
- Privacy (Privasi) merujuk pada hak individu untuk mengontrol bagaimana informasi pribadi mereka dikumpulkan, digunakan, dan dibagikan, bahkan oleh pihak yang memiliki akses yang sah (authorized access).
- Keamanan yang baik adalah fondasi untuk privasi. Kita tidak bisa memiliki privasi tanpa keamanan. Tapi, kita bisa punya privasi yang buruk walau dilengkapi keamanan yang bagus.

**Referensi:** [Kind of a stupid question, but are Privacy controls different than security controls? : r/NISTControls](#)

##### b. Vulnerability vs Threat

- **Vulnerability (Kerentanan)** adalah **kelemahan atau celah** dalam sistem, proses, atau kontrol yang dapat dieksloitasi. Kerentanan sendiri bersifat **pasif**; ia adalah sebuah **kondisi**.
- **Threat (Ancaman)** adalah **aktor atau peristiwa** yang berpotensi mengeksloitasi sebuah kerentanan untuk menyebabkan kerugian. Ancaman bersifat **aktif**; ia adalah sesuatu yang **bisa terjadi**.
- Sebuah kerentanan tidak akan berbahaya jika tidak ada ancaman yang bisa mengeksloitasinya. Sebaliknya, sebuah ancaman tidak akan berhasil jika tidak ada kerentanan yang bisa dimanfaatkan. Risiko keamanan muncul ketika sebuah **ancaman** bertemu dengan sebuah **kerentanan**.

**Referensi:**

- <https://csrc.nist.gov/glossary/term/vulnerability>
- <https://csrc.nist.gov/glossary/term/threat>

c. *Incident vs Attack*

- **Attack (Serangan)** adalah **tindakan yang disengaja** oleh seorang aktor ancaman untuk mencoba melewati kontrol keamanan dan merusak, mengubah, atau mencuri informasi. Serangan adalah **upaya (buruk)**
- **Incident (Insiden)** adalah **peristiwa** di mana keamanan sistem atau data **telah atau berpotensi terganggu**. Sebuah insiden adalah **akibat** atau **kejadian** yang dikonfirmasi, yang bisa jadi hasil dari serangan yang berhasil, atau bahkan dari kesalahan yang tidak disengaja.
- Sebuah serangan adalah salah satu kemungkinan penyebab dari sebuah insiden. Tidak semua serangan akan menjadi insiden (jika serangan berhasil digagalkan), dan tidak semua insiden disebabkan oleh serangan.

**Referensi:**

- <https://csrc.nist.gov/glossary/term/attack>
- <https://csrc.nist.gov/glossary/term/incident>

d. *Authentication vs Authorization*

- **Authentication (Autentikasi)** adalah proses untuk **memverifikasi identitas** seseorang atau sesuatu. Ini menjawab pertanyaan, "Siapa Anda?"
- **Authorization (Otorisasi)** adalah proses untuk **menentukan hak akses** yang dimiliki oleh identitas yang telah diautentikasi.
- Autentikasi dan otorisasi adalah dua proses yang saling melengkapi dan biasanya terjadi secara berurutan.
  1. **Pertama, Autentikasi.** Seseorang harus membuktikan siapa diri terlebih dahulu. Contoh: Seseorang *login* ke sistem perbankan dengan username dan kata sandi.
  2. **Kemudian, Otorisasi.** Setelah sistem tahu siapa seseorang , ia akan memeriksa daftar izin orang tersebut untuk menentukan apa yang bisa dilihat dan dilakukan.

**Referensi:** [Authentication vs. Authorization: What's the Difference?](#) | [OneLogin](#)

2. **(2 poin)** Sebuah security operations center menggunakan Wazuh, Suricata, Kafka, Elasticsearch, dan Kibana. Jelaskan peran masing-masing teknologi serta aliran data yang ada!
3. **(4 poin)** Yayasan Rumah Perapian (YRP) sedang mempersiapkan sebuah sistem SSO untuk seluruh aplikasi internalnya. Buat sebuah kebijakan keamanan informasi yang berisi kebijakan, standar, prosedur, dan *guideline* terkait (kewajiban) login menggunakan SSO!
4. **(10 poin)** PT Inazuma Bersinar Selamanya (IBS) menyediakan berbagai layanan dalam bidang kelistrikan dan teknologi informasi. Untuk meningkatkan efisiensi, IBS membuat sebuah aplikasi web yang menyediakan portal interaksi terpusat bagi manajemen IBS, mitra kerja IBS, dan klien IBS.
  - a. Buat *threat model*/keamanan informasi untuk aplikasi web tersebut.
    - i. Identifikasi seluruh asset dan aktor yang mungkin terlibat dengan aplikasi!
    - ii. Menggunakan metodologi STRIDE, identifikasi kemungkinan ancaman-ancaman terhadap aplikasi!

- b. Aplikasi dikembangkan tanpa melibatkan keamanan informasi secara menyeluruh. Setelah aplikasi diluncurkan, sekelompok *hacker* dari Snezhnaya berhasil mendapatkan akses tertinggi ke aplikasi. Penyelidikan menemukan bahwa para *hacker* mendapatkan akses awal dengan melakukan gabungan antara serangan *social engineering* dan *brute force* untuk mendapatkan akses ke akun mitra, sebelum kemudian memanfaatkan *endpoint API* tertentu dengan otentikasi lemah untuk menaikkan tingkat akses.
- i. Sebutkan dan jelaskan tiga kategori OWASP Top 10:2021 yang relevan terhadap kasus di atas!
  - ii. Jelaskan bagaimana Secure SDLC dapat mencegah terjadinya kasus tersebut, serta kegiatan-kegiatan yang relevan pada tiap tahap!
  - iii. Sebutkan dan jelaskan setidaknya satu contoh kasus atau insiden keamanan yang serupa. Berikan kronologi kasus tersebut beserta dengan pelajaran-pelajaran yang dapat digunakan pada kasus IBS; jangan lupa untuk juga mencantumkan referensi-referensi yang digunakan.

## V. Miscellaneous



If you don't do these, he'll haunt your dreams tonight

1. **(10.0 poin - WAJIB)** Sebagai asisten, nantinya kamu akan dituntut untuk memastikan seluruh dan segala bentuk kecurangan akademis dalam mata kuliah-mata kuliah yang diasistensi ditemukan, dilaporkan, dan ditindaklanjuti.

- a. Sebutkan landasan (bebas; hukum, agama, etika, dan seterusnya) yang mewajibkan asisten untuk melakukan hal tersebut.

- Peraturan akademik STEI (kena DO euy ngerinyaaaa)
- Kata hati (katanya jangan mempertahankan orang yang curang)
- Pancasila dan Kebenaran Ilmiah (#real)

- b. Sebutkan tiga alat yang mampu membantumu menemukan dan atau mengonfirmasi tindak laku kecurangan.

Ni kita bedain alatnya berdasarkan konteksnya aja ya

- Cek laporan: Sebenarnya bukan pakai alat tambahan, tapi bakal minta submit versi docs (via link aja). Kalau ternyata dia make laporan orang tapi di rename, harusnya keliatan. Kalau mau pdf, bisa pakai TurnItIn

- Cek metadata: ya langsung aja si, pake bawaan. Bisa aja keliatan soalnya nama asli yang punya file
- Kalau **praktikum luring (sinkron)**, kita bisa pakai Keylogger (biar keliatan apakah dia beneran ngetik atau ternyata buka yang lain-lain)

c. Jelaskan garis yang akan kamu tarik terkait kecurangan dan plagiarisme.

Disini aku bakal bedain plagiarisme. Menurut aku ada plagiarisme ringan dan plagiarisme berat (yang termasuk ke ranah curang)

#### **Ringan:**

- Ngambil ide orang atas nama “inspirasi”, tapi ga dikembangkan lagi
- Ngambil sumber dari orang lain, diederit lagi, tapi ga nyantumin sumber

Menurut aku ini orangnya bisa dibina lagi biar emg paham kalo hal-hal ini masuk plagiarisme

#### **Berat:**

- copas kode dari Stack Overflow, GitHub, blog, AI, tanpa menyebutkan sumbernya di dalam komentar kode atau laporan. Minimal bilang aja jir.
- Ngambil tugas milik orang lain dari bahasa asing dan mengakuinya sebagai karya sendiri.

Sebenarnya ini udah masuk curang sih. Mereka melangkahi kerja keras orang untuk nilai yang sama.

#### **Literally curang:**

- Joki
- Nyontek pas ujian dan praktikum (tp tergantung tipenya apa)
- Pemalsuan data
- Resubmit tugas dari matkul lain (kalau ternyata topiknya sama)

d. Berikut merupakan beberapa studi kasus. Untuk masing-masing kasus, jelaskan dengan singkat apa saja yang akan kamu lakukan untuk menentukan verdict akhir.

i. Terdapat kelompok tugas besar yang kodennya sangat mirip dengan kode yang tersedia di internet, namun mereka tidak memberikan pengakuan (credits) atasnya.

Karena dibilang sangat *mirip*, aku akan cek kode kelompoknya dan bandingin sama kode di internet tsb, paling pakai MOSS atau manual kalau kuat. Kalau indikasinya kuat, nilai 0 untuk kelompok tsb dan pelaporan kepada dosen terkait

ii. Terdapat kelompok tugas besar yang kodennya sangat mirip dengan kode tugas besar karya kelompok kakak tingkat.

Sama sih, karena dibilang sangat *mirip*, aku akan cek kode kelompoknya dan bandingin sama kode di internet tsb, paling pakai MOSS atau manual kalau kuat. Terus juga karena ini kaitannya sama katingnya, kayaknya aku akan wawancara personal perwakilan kelompok tsb, kemudian wawancara kating tsb.

Kalau indikasinya kuat, nilai 0 untuk kelompok tsb dan pelaporan kepada dosen terkait.

iii. Terdapat beberapa mahasiswa yang saling berbincang dan tengok menengok ketika ujian.

Palingan aku samper, dan kalau terlihat sekilas jawabannya sangat *mirip*, akan dicatat di berita acara.

iv. Terdapat beberapa mahasiswa yang jawabannya sangat mirip satu sama lain, dan jawaban tersebut bukanlah jawaban standar.

Gabis langsung tindak kalo ujian/prak, emang siapa tau orang-orang belajar bareng dan itu yang mereka pelajarin semalem.

Harus emang nanya satu per satu, setelah ujian/prak/dkk. Definisi jawaban standar juga beda-beda kan untuk tiap orang.

- v. Terdapat mahasiswa yang jawabannya sangat mirip dengan mahasiswa lain yang kamu ketahui sering memberikan tutor, dan bukan merupakan seseorang yang sepertinya akan melakukan kecurangan.

Tetep cek dua duanya. Dua-duanya kita tetep pegang praduga tak bersalah ya. Kita harus cek berbagai sumber yang mungkin, disini berarti bisa aja ada video, dokumen, dkk. Balik lagi nih, kalo ternyata tutornya yang nyontek gimana? Atau ternyata gada yang nyontek? Nah itu makanya harus cek semua sumber yang ada. Tanya langsung kalo bisa

- vi. Terdapat mahasiswa yang terbukti, berdasarkan sebuah detail yang sepertinya lupa diganti, mengumpulkan jawaban temannya.

Jujur kasian sih ini orang. Minimal kalau ga knowledge-issue, ga skill-issue. Tapi yaudah, jelas di O in :)

- e. Apakah kamu akan meresikokan reputasi personal, hubungan-hubungan interpersonal, maupun kestabilan organisasi non-akademik (seperti himpunan atau unit) untuk memastikan integritas akademik?

Integritas akademik itu susah dibangun. Lebih baik memperkuat dan membudayakan integritas akademik daripada takut akan reputasi dan relasi. Relasi bisa dibangun lagi, setidaknya dan seminimalnya dengan orang yang berbeda dan sevisi. Tapi, pelanggaran integritas akademik akan selalu melekat. Lagipula, kampus ini tujuannya untuk belajar dan kembali memberi untuk masyarakat. Jikalau tujuan awalnya aja udah loss, ngapain masih kuliah? Kalau emang waras/sehat, organisasi/hubungan yang sehat tsb harusnya ga mempermasalahkan adanya penegakan integritas akademik kok

**Final Verdict: Siap.**