UNIVERSITAS INDONESIA

PEERTOCP: EDITOR KODE DAN *SHARED SHELL* KOLABORATIF DALAM WAKTU NYATA BERBASIS WEBRTC

SKRIPSI

HOCKY YUDHIONO
1906285604

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
DESEMBER 2022

UNIVERSITAS INDONESIA

PEERTOCP: EDITOR KODE DAN *SHARED SHELL* KOLABORATIF DALAM WAKTU NYATA BERBASIS WEBRTC

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar

Sarjana Ilmu Komputer

HOCKY YUDHIONO

1906285604

FAKULTAS ILMU KOMPUTER

PROGRAM STUDI ILMU KOMPUTER

DEPOK

DESEMBER 2022

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Hocky Yudhiono**
NPM	:	**1906285604**
Tanda Tangan	:	
Tanggal	:	**1 Desember 2022**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Hocky Yudhiono
NPM	:	1906285604
Program Studi	:	Ilmu Komputer
Judul Skripsi	:	PeerToCP: Editor Kode dan *Shared Shell* Kolaboratif dalam Waktu Nyata Berbasis WebRTC

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing 1	:	Muhammad Hafizhuddin Hilman S.Kom., M.Kom., Ph.D.	( )
Penguji 1	:	Penguji Pertama Anda	( )
Penguji 2	:	Penguji Kedua Anda	( )

Ditetapkan di	:	Depok
Tanggal	:	1 Desember 2022

KATA PENGANTAR

Puji syukur kita panjatkan ke hadirat Tuhan Yang Maha Esa, karena rahmat dan anugerah-Nya, penulis dapat menyelesaikan skripsi yang berjudul "PeerToCP: Editor Kode dan *Shared Shell* Kolaboratif dalam Waktu Nyata Berbasis WebRTC" yang menjadi salah satu syarat kelulusan dalam menempuh pendidikan Sarjana Ilmu Komputer di Fakultas Ilmu Komputer, Universitas Indonesia. Penulis juga ingin berterima kasih kepada pihak-pihak lain, khususnya kepada:

- Bapak Muhammad Hafizhuddin Hilman S.Kom., M.Kom., Ph.D. selaku dosen pembimbing tugas akhir yang senantiasa melakukan supervisi kepada penulis dan memberikan pengetahuan setiap pekannya;
- Ibu Dr. Putu Wuri Handayani, S.Kom., M.Sc., Ibu Dipta Tanaya, S.Kom., M.Kom., dan Ibu Dr. Eng. Laksmita Rahadianti S.Kom., M.Sc. selaku dosen yang mengarahkan penulis dalam ilmu metodologi penelitian dan penulisan ilmiah;
- seluruh dosen yang dengan sabar mengajarkan ilmunya selama menempuh studi di Fakultas Ilmu Komputer, Universitas Indonesia;
- kedua orang tua, keluarga, dan kakak-kakak penulis yang mendukung proses perkuliahan sembari menyelesaikan skripsi ini;
- serta teman-teman penulis: Eko, Joni, Steven Wiryadinata, Samuel, Kak Prabowo, Pikatan, Kenta, Andre, Irfancen, Raihan, Rafi, Aimar, Lucky, Fausta, Novaryo, Kak Rey, dan teman-teman lain yang tidak dapat penulis sebutkan satu per satu namanya, karena setia menemani dan memberikan dukungan mental kepada penulis.

Penulis juga menyadari bahwa masih terdapat kesalahan dan kekurangan dalam penulisan karya ilmiah ini. Penulis berharap karya tulis ini dapat memberikan manfaat dan inspirasi untuk pengembangan dan peradaban ilmu pengetahuan teknologi dan informatika dunia, terutama bangsa Indonesia.

Depok, 1 Desember 2022

Hocky Yudhiono

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Hocky Yudhiono
NPM	:	1906285604
Program Studi	:	Ilmu Komputer
Fakultas	:	Ilmu Komputer
Jenis Karya	:	Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty Free Right*)** atas karya ilmiah saya yang berjudul:

PeerToCP: Editor Kode dan *Shared Shell* Kolaboratif dalam Waktu Nyata Berbasis WebRTC

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 1 Desember 2022
Yang menyatakan

(Hocky Yudhiono)

ABSTRAK

Nama	:	Hocky Yudhiono
Program Studi	:	Ilmu Komputer
Judul	:	PeerToCP: Editor Kode dan *Shared Shell* Kolaboratif dalam Waktu Nyata Berbasis WebRTC
Pembimbing	:	Muhammad Hafizhuddin Hilman S.Kom., M.Kom., Ph.D.

Aplikasi kolaboratif dalam waktu nyata menjadi bagian dari kehidupan manusia modern saat ini. Teknologi ini digunakan terutama untuk berkomunikasi dan meningkatkan produktivitas dengan mengurangi hambatan ruang dan waktu. Salah satu sistem aplikasi yang marak digunakan adalah editor dokumen yang dapat digunakan beberapa pengguna secara bersamaan. Dilatar belakangi oleh motivasi tersebut, penelitian ini memaparkan sebuah aplikasi editor kode kolaboratif *local-first* berbasis *peer-to-peer* yang diimplementasi dengan WebRTC dan CRDT. Selain itu, aplikasi ini menyertai *shell* bersama yang dapat dijalankan oleh salah satu pengguna dan digunakan oleh setiap pengguna lain dalam suatu kelompok jaringan. Terdapat beberapa variasi arsitektur *backend* pada aplikasi yang dibandingkan dalam penelitian ini. Dari segi algoritma dalam menjaga konsistensi dokumen, dua pendekatan berbeda yang diteliti yakni algoritma OT (*operational transformation*) dan metode yang memanfaatkan struktur data CRDT (*conflict-free replicated data types*). Dari segi arsitektur jaringan, penelitian ini mengevaluasi CRDT berbasis *client-server*, CRDT berbasis *peer-to-peer*, serta OT berbasis*client-server*. Keterbatasan OT yang diimplementasi pada penelitian ini membutuhkan suatu sumber kebenaran berupa server, sehingga OT berbasis *peer-to-peer* tidak dievaluasi. Penelitian ini menemukan bahwa variasi implementasi CRDT *peer-to-peer* yang diujikan memiliki performa lebih baik untuk sejumlah pengguna $n \leq 8$. Selain itu, *signalling server* pada variasi ini menggunakan *resource* yang minim, sehingga lebih optimal untuk kelompok jaringan yang lebih banyak. Terlepas dari hal tersebut, variasi CRDT *client-server* memiliki pertumbuhan kompleksitas waktu yang lebih rendah terhadap jumlah pengguna dalam satu kelompok jaringan dan penggunaan *resource* pada sistem klien yang lebih rendah, namun lebih tinggi pada servernya. Sehingga variasi CRDT *client-server* dapat dipertimbangkan penggunaannya ketika terjadi masalah saat melakukan inisialiasi jaringan *peer-to-peer* atau jumlah pengguna dalam suatu kelompok jaringan jauh lebih banyak dari eksperimen yang dilakukan pada penelitian ini.

Kata kunci:
WebRTC, WebSocket, *local-first*, waktu nyata, editor kode, *conflict-free replicated data types*, *operational transformation*, *shell* bersama, Yjs, Electron

ABSTRACT

Name	:	Hocky Yudhiono
Study Program	:	Computer Science
Title	:	PeerToCP: WebRTC Based Real-Time Collaborative Code Editor and Shared Shell
Counsellor	:	Muhammad Hafizhuddin Hilman S.Kom., M.Kom., Ph.D.

Real-time collaborative applications are becoming a part of modern human life today. These technologies are used primarily to communicate and increase productivity by reducing time and space barriers. One of the widely used application systems is a document editor that can be used by multiple users simultaneously. Based on this motivation, this research presents a peer-to-peer and local-first collaborative code editor application implemented with WebRTC and CRDT. In addition, the application includes a shared shell that can be run by one user and used by every other user in a network group. There are several variations of backend architecture in the applications compared in this study. In terms of algorithms for maintaining document consistency, two different approaches were evaluated, namely OT (operational transformation) algorithm and CRDT (conflict-free replicated data types) data structure. In terms of network architecture, this study assessed client-server based CRDT, peer-to-peer based CRDT, and client-server based OT. The limitation of OT implemented in this research is that it requires a single source of truth in the form of a server, so peer-to-peer-based OT was not evaluated. This study found that the peer-to-peer based CRDT variation tested performed better for a number of users $n \leq 8$. Moreover, the signaling server in this variation uses minimal resources, making it more optimal for larger network groups. However, the client-server CRDT variation has lower time complexity growth with respect to the number of users in a network group and lower resource usage on the client system, but higher on the server. Therefore, the client-server CRDT variation's usage can be considered when there are problems initializing a peer-to-peer network or the number of users in a network group is much larger than the experiments conducted in this study.

Key words:
WebRTC, WebSocket, local-first, real-time, code editor, conflict-free replicated data types, operational transformation, shared shell, Yjs, Electron

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB 1
PENDAHULUAN

Bab ini membahas latar belakang perkembangan teknologi komunikasi waktu nyata, aplikasinya di internet pada saat ini, serta tantangan dan hambatan perkembangan teknologi ini untuk dapat digunakan secara komersial bagi publik. Melalui latar belakang, disusun gagasan pengembangan dan implementasi sederhana sistem aplikasi yang memanfaatkan teknologi tersebut. Ada pula tujuan, manfaat, serta batasan penulisan untuk memberikan konteks visi, misi, dan lingkup pengembangan yang turut disampaikan pada bab ini.

1.1 Latar Belakang

Penggunaan aplikasi yang menghubungkan manusia, baik secara lisan maupun tulisan secara waktu nyata (*real-time*) semakin marak digunakan. Pada situasi pandemi COVID-19 semasa penelitian ini, perusahaan dan institusi pendidikan mengadakan aktivitas jarak jauh dan membutuhkan suatu media komunikasi yang dapat diandalkan. Beberapa aplikasi yang umum digunakan ialah produk-produk Google Workspace, seperti Google Docs, Google Slides, Google Meet, ataupun dari pengembang lain, yakni WhatsApp, LINE, JetBrains Code With Me, Zoom, dan masih banyak lagi. Tidak lepas pula dari permainan video multipemain yang membolehkan pemainnya berinteraksi secara langsung dengan latensi yang cenderung rendah dan terasa seperti waktu nyata.

Teknologi komunikasi waktu nyata atau RTC (*real-time communications*) merupakan sebuah istilah metode telekomunikasi untuk beberapa pengguna yang berinteraksi dengan latensi atau waktu jeda yang relatif rendah terhadap respons pengguna (Arefin, Azad, & Kabir, 2013). Teknologi RTC mulai menjadi fokus penelitian dan penggunaan sejak dikenalkannya teknologi WebSocket pada tahun 2008 (Fette & Melnikov, 2011; Reynolds, 2008). Penggunaan WebSocket dimanfaatkan untuk menurunkan latensi dan membuat komunikasi dalam waktu nyata dapat terwujud melalui implementasi yang optimal.

RTC (*real-time communications*) dengan WebSocket diaplikasikan pada salah satu aplikasi penyuntingan dokumen yang umum digunakan yakni Google Docs yang fiturnya dikembangkan pada tahun 2010 (Belomestnykh, 2010). Google Docs menggunakan metode khusus yaitu OT (*operational transformation*) yang mendukung berbagai kapa-

bilitas kolaborasi (Day-Richter, 2010; Harris, 2010). Teknologi ini mulai berkembang dan diteliti pada tahun 1989 (Ellis & Gibbs, 1989). Sepanjang perkembangannya, ada beberapa isu kesalahan pada metode OT yang terdeteksi dan diselesaikan secara bertahap. Implementasi dari metode ini juga memiliki banyak variasi serta keuntungan dan kerugiannya masing-masing, baik dari aspek memori maupun waktu. Salah satu pengembang aplikasi Google Wave, yang merupakan teknologi pendahulu Google Docs membutuhkan waktu sekitar dua tahun untuk menyelesaikan implementasi dari metode OT ini (Gentle, 2011). Meskipun membutuhkan waktu lama, Google Docs menjadi salah satu produk editor teks andalan dengan kolaborasi waktu nyata yang memanfaatkan arsitektur *client-server* dan masih digunakan hingga saat ini.

Seiring perkembangan teknologi, pada tahun 2011 WebRTC dikenalkan sebagai protokol dan antarmuka pemrograman aplikasi yang mendukung komunikasi waktu nyata dua arah yang bekerja secara *peer-to-peer* (Dutton et al., 2012). WebRTC menyediakan suatu protokol untuk membolehkan suatu pengguna untuk berkomunikasi langsung dengan setiap pengguna lain tanpa melalui server setelah melakukan proses *signalling*. *Signalling* merupakan istilah untuk melakukan inisiasi koneksi kanal data WebRTC melalui sebuah server (Sredojev, Samardzija, & Posarac, 2015). WebRTC dikembangkan dengan tujuan utama untuk melakukan komunikasi waktu nyata dengan data yang lebih besar, seperti media suara dan video (Dutton et al., 2012). Muatan ke server juga menjadi lebih ringan karena hanya digunakan untuk *signalling*. Hal ini cenderung meningkatkan skalabilitas dan menyediakan lebih banyak ketersediaan jaringan WebRTC terhadap pengguna bila dibandingkan dengan *client-server*.

WebRTC juga mulai diteliti untuk dapat digunakan dalam berbagai kasus penggunaan, salah satunya untuk penyuntingan dokumen secara berkolaborasi dan dalam waktu nyata. Metode OT yang umumnya digunakan pada arsitektur *client-server* memiliki beberapa properti khusus pada sifat konvergensi hasil akhirnya yang menyebabkan implementasinya pada arsitektur *peer-to-peer* akan lebih sulit (C. Sun, Xu, & Ng, 2017). Sesuai namanya, OT (*operational transformation*) meresolusi dengan melakukan transformasi terhadap operasi-operasi penyuntingan yang dilakukan terhadap suatu dokumen (Smith, 2012). Pada arsitektur *client-server*, metode ini mengandalkan suatu server sebagai satu sumber kebenaran data (*single source of truth*) yang akan menyelesaikan resolusi setiap operasi yang masuk dari setiap kliennya.

Perkembangan struktur data yang disebut dengan *conflict-free replicated data types*

(CRDT) mulai menjadi alternatif untuk metode resolusi pada arsitektur *peer-to-peer*. CRDT dikenalkan pada tahun 2006 dan mulai secara formal didefinisikan pada tahun 2011 (Shapiro, Preguiça, Baquero, & Zawirski, 2011). Struktur data ini digunakan pada komputasi sistem terdistribusi dan tidak membutuhkan koneksi yang selalu tersedia setiap saatnya bagi semua pengguna. Pada CRDT, resolusi dilakukan pada *state* atau kondisi dokumen saat ini dan tidak melalui transformasi dari operasi-operasi penyuntingannya (Preguiça, 2018). CRDT dapat pula digunakan pada arsitektur *client-server*, dengan setiap resolusi diselesaikan pada server, dan perubahan akan di-*broadcast* atau disebarkan pada setiap klien (C. Sun, Sun, Agustina, & Cai, 2019).

Kedua arsitektur, yakni *client-server* dan *peer-to-peer* memiliki banyak keuntungan dan kerugian. Reliabilitas dan sumber daya yang terfokus pada server dapat menjadi terbatas, namun lebih stabil karena performanya yang dipersiapkan oleh pengembang. Di lain sisi, arsitektur *peer-to-peer* dipertimbangkan karena mengurangi waktu *overhead* dalam berkomunikasi antarpenggunanya karena berhubungan langsung dan tidak melalui server. Jaringan *peer-to-peer* dapat bekerja secara optimal saat banyaknya pengguna dalam suatu jaringan kecil (Leibnitz, Hoßfeld, Wakamiya, & Murata, 2007; Maly, Mischke, Kurtansky, & Stiller, 2003). Namun sebaliknya, jumlah koneksi dalam jaringan akan meningkat dengan kompleksitas $O(N^2)$ dengan susunan *mesh* bila setiap pengguna terhubung dengan pengguna lainnya. Hal ini menyebabkan komunikasi data yang dilakukan akan meningkat dalam waktu kuadratik dan tidak efisien karena transmisi untuk data yang sama dilakukan untuk semua pengguna. Terlepas dari kelebihan dan kekurangan yang disampaikan, beberapa aplikasi editor kode kolaboratif waktu nyata yang ada saat ini dikembangkan dengan arsitektur tertentu sesuai dengan kebutuhannya. Misalnya terdapat editor kode Atom dengan *plugin* Teletype, Brackets, dan JetBrains Code With Me yang berbasis *peer-to-peer*, atau pun codecollab, codeshare, dan Google Docs yang berbasis *client-server*.

Editor kode kolaboratif lebih lanjut dapat dikembangkan menjadi suatu *environment* yang membolehkan kolaborasi untuk digunakan dalam pemrograman kompetitif. Pemrograman kompetitif merupakan cabang olahraga pemrograman yang diperlombakan secara individu atau berkelompok untuk mengerjakan soal komputasional dengan batasan waktu dan memori tertentu. Pada pemrograman kompetitif, kode yang diedit umumnya bersifat sebuah berkas tunggal (*single file*), yang dapat dikompilasi atau dijalankan tersendiri. Beberapa bahasa yang umum digunakan antara lain C, C++, Python, dan Java. Penelitian ini

akan membahas pengembangan sebuah aplikasi editor kode, akses kompilator, dan *shell* kolaboratif yang mendukung fitur pemrograman bersama dalam waktu nyata untuk setiap pengguna dalam sebuah jaringan. Penelitian ini juga menunjukkan hasil analisis *benchmarking* performa PeerToCP yang menggunakan metode resolusi *operational transformation* berbasis *client-server*, struktur data CRDT berbasis arsitektur *peer-to-peer*, serta struktur data CRDT pula, namun berbasis *client-server*.

1.2 Pertanyaan Penelitian

Berdasarkan paparan latar belakang dan rumusan masalah yang disampaikan, berikut adalah pertanyaan-pertanyaan yang hendak dijawab dari penelitian.

1. Bagaimana implementasi dari beberapa variasi aplikasi PeerToCP yang menyediakan *real-time collaborative code editor* dan *shared shell*?
2. Bagaimana perbandingan performa, latensi, dan kebutuhan *resource* sistem pengguna dan server PeerToCP berbasis WebRTC dengan variasinya yang lain?
3. Berdasarkan hasil analisis dan evaluasi, apa keuntungan PeerToCP dengan arsitektur *peer-to-peer* dan bagaimana pertimbangan penggunaan sistem dengan variasi tertentu?

1.3 Batasan Penelitian

Pertanyaan penelitian yang disampaikan pada subbab sebelumnya dibatasi oleh beberapa batasan penelitian. Pembatasan ini untuk memberikan kejelasan cakupan dan jangkauan penelitian yang disampaikan dalam karya ini. Dalam penelitian ini, *user interface* dan *user experience* dari bagian tampilan aplikasi tidak akan diuji dengan metode interaksi manusia dan komputer. Evaluasi pada penelitian ini juga diujikan dengan asumsi koneksi dan jaringan internet yang tidak terkendala, serta jumlah pengguna yang terbatas seperti pemrograman kompetitif pada umumnya. PeerToCP masih menggunakan *library*, modul, dan *framework* yang sudah tersedia dengan modifikasi dan penyesuaian seperlunya dalam proses pengembangan sistem aplikasi. Untuk setiap variasi PeerToCP, terdapat beberapa perbedaan fitur dan perilaku minor (tidak signifikan terhadap operasi yang akan dievaluasi) yang diabaikan.

1.4 Tujuan Penelitian

Terlepas dari batasan dan cakupannya, penelitian ini bertujuan untuk mewujudkan potensi dari teknologi *real-time communications* yakni WebSocket dan WebRTC dalam bentuk aplikasi PeerToCP, sebuah *environment* pemrograman kompetitif kolaboratif *real-time*. Bersama tujuan tersebut, detail implementasi dipaparkan secara detail untuk menerangkan hambatan dan solusi yang diambil dalam pengembangan aplikasi ini. Penelitian ini juga menunjukkan evaluasi perbandingan variasi implementasi dari PeerToCP dengan uji skenario berbagai operasi esensial yang dapat dilakukan.

1.5 Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan gambaran umum dan bentuk prototipe dasar beberapa teknologi *real-time communication*, seperti teknologi WebRTC (*Web Real-Time Communication*) dan beberapa metode resolusi sinkronisasi replika data dalam beberapa pengguna dalam sebuah jaringan. Lebih lanjut, aplikasi ini berpotensi untuk dikembangkan lebih lanjut ke tahap produksi dan digunakan secara komersial. Penelitian ini juga didesain untuk menjadi acuan dalam penelitian tolok ukur lain terkait dengan performa arsitektur yang digunakan terhadap beberapa operasi komunikasi data tertentu, terutama dalam bentuk kolaborasi penyuntingan teks dan sinkronisasi data waktu nyata. Berangkat dari tujuan penelitian yang disampaikan sebelumnya pula, beberapa permasalahan dan hambatan pengembangan yang dipaparkan dapat diteliti lebih lanjut untuk membuat alternatif solusi yang lebih optimal terhadap solusi yang disampaikan pada penelitian ini.

1.6 Sistematika Penulisan

Untuk memberikan konteks penelitian yang padu dan terurut, laporan penelitian yang disampaikan dalam karya ini dibagi menjadi enam bagian, antara lain sebagai berikut.

1. Bab I Pendahuluan, memberikan konteks dasar dan pendahuluan dari penelitian, termasuk latar belakang, rumusan masalah yang terdiri dari pertanyaan penelitian dan batasannya, tujuan dan manfaat penelitian, serta sistematika penulisan keseluruhan tulisan.

2. Bab II Tinjauan Pustaka, menyampaikan dasar-dasar studi dari pustaka yang berhubungan dengan penelitian yang dilakukan. Bab ini juga memberikan pengertian terminologi, teori, dan konsep tertulis terkait.
3. Bab III Metodologi Penelitian, menerangkan metodologi yang digunakan dalam penelitian ini termasuk tahapan penelitian, desain implementasi, skenario pengujian, dan metrik evaluasi.
4. Bab IV Implementasi, membahas detail implementasi dari aplikasi PeerToCP dengan berbagai variasinya.
5. Bab V Hasil dan Pembahasan, menjelaskan hasil evaluasi terhadap pengujian performa yang dilakukan.
6. Bab VI Penutup, memberikan kesimpulan serta saran akhir untuk perkembangan penelitian selanjutnya.

BAB 2
TINJAUAN PUSTAKA

Untuk menjawab pertanyaan penelitian yang diuraikan pada bab 1, dibutuhkan dasar pengetahuan yang sesuai. Informasi ini berguna untuk mengetahui potensi pengembangan aplikasi dari berbagai tulisan dan penelitian sebelumnya. Secara umum, bab ini memaparkan mengenai teknologi-teknologi yang terkait dengan pengembangan aplikasi, antara lain WebRTC, CRDT (*conflict-free replicated data types*), OT (*operational transformation*), dan sifat-sifat pada sebuah editor teks, terutama untuk editor kode. Bab ini juga akan memberikan gambaran mengenai penelitian terkait dan sistem-sistem aplikasi yang sudah pernah dikembangkan sebelumnya. Dalam penelitian ini, terdapat banyak teknologi yang digunakan sebagai media transmisi data, salah satu solusinya dalam aplikasi yang berbasis *client-server* adalah teknologi WebSocket.

2.1 WebSocket

WebSocket merupakan protokol komunikasi dengan kanal dua arah, atau biasa dikenal dengan *full-duplex* yang diinisiasi melalui sebuah koneksi TCP (*Transmission Control Protocol*) (Fette & Melnikov, 2011). Protokol ini bersifat *stateful*, yang berarti koneksi antara klien dan server akan terus bertahan hingga salah satu pihak memutuskan hubungannya (Pimentel & Nickerson, 2012). Pada arsitektur perangkat lunak, teknologi ini dapat dimanfaatkan untuk membuat sebuah sistem dengan *publisher-subscriber design pattern* (Ganaputra & Pardamean, 2015). Pada pola ini, klien dapat melakukan permintaan berlangganan ke suatu server dan menjalin hubungan WebSocket. Sementara server akan senantiasa memberikan arus data terus-menerus setelah adanya pembaruan kepada setiap klien yang berlangganan. Selain itu, protokol RPC (*Remote Procedure Call*) juga dapat diterapkan di atas WebSocket. RPC merupakan istilah pada sistem terdistribusi yang bekerja seperti pemanggilan fungsi pada sebuah *service* atau layanan aplikasi dengan parameter tertentu (Srinivasan, 1995). Pada WebSocket, setiap pemanggilan *request* RPC menggunakan kanal komunikasi yang sama dan sudah tersedia, sehingga memberikan latensi yang jauh lebih optimal tanpa biaya inisiasi awal tambahan.

WebSocket merupakan teknologi yang sudah ada selama lebih dari 13 tahun saat

penelitian ini dilakukan (Fette & Melnikov, 2011). Berbagai protokol lain kian diteliti untuk mengoptimalkan abstraksi teknologi ini, yaitu dengan mempertahankan fiturnya dan mempercepat performanya. Secara umum, karakteristik WebSocket berbeda dari protokol HTTP atau HTTPS yang bersifat *stateless*. Namun, HTTP/3.0 memberikan potensi protokol baru yang dapat menggantikan WebSocket. Perkembangannya diawali dari HTTP/1.1 yang merupakan salah satu versi protokol HTTP yang dikenalkan pada awal tahun 1997 dan masih digunakan hingga saat ini (Fielding & Reschke, 2015; Krishnamurthy, Mogul, & Kristol, 1999). Pada protokol HTTP/1.1, koneksi TCP yang mendasarinya dapat dipertahankan (*persisted*) dan setiap permintaan atau *request* dikirimkan satu per satu secara berurutan tanpa membuat koneksi baru. Koneksi akan ditutup setelah semua *request* HTTP/1.1 selesai diterima (Fielding & Reschke, 2015).

Perkembangan HTTP dilanjutkan oleh HTTP/2.0, versi HTTP yang dipublikasikan pada tahun 2015 dan menyediakan fitur *multiplexing*. Setiap *request* pada protokol ini dapat dikirimkan secara paralel dalam suatu koneksi TCP dan membolehkan transmisi data yang lebih efektif (Belshe, Peon, & Thomson, 2015). Secara teori, WebSocket dapat digantikan oleh HTTP/2.0 yang menyediakan *stream full-duplex* (Stenberg, 2014). Namun, *multiplexing* dan persistensi koneksi pada HTTP/2.0 ditujukan bukan sebagai pengganti WebSocket (Fietze, 2017). Versi HTTP/3.0 yang secara teknis berbasis UDP (*User Datagram Protocol*) menyediakan API (*application programming interface*) yang dikenal dengan WebTransport sebagai alternatif dari WebSocket yang lebih optimal (Bishop, 2022; Frindell, Kinnear, & Vasiliev, 2022). Hingga waktu penelitian ini dilakukan, protokol *WebTransport* masih dalam pengembangan dan bersifat *draft*. Penggunaannya juga bersifat eksperimental dan terbatas untuk *browser* tertentu (Frindell et al., 2022). Teknologi WebSocket digunakan pada layanan yang berbasis *client-server*. Terdapat beberapa protokol lain yang didesain untuk digunakan dalam menghubungkan klien secara langsung atau lebih dikenal dengan arsitektur *peer-to-peer*, salah satunya ialah WebRTC.

2.2 WebRTC

WebRTC merupakan sebuah teknologi web pada browser dan perangkat telepon yang membolehkan koneksi langsung berbasis *peer-to-peer* dalam transmisi datanya (Alvestrand, 2021a). WebRTC tidak hanya API (*Application Programming Interface*), namun juga termasuk protokol yang telah didefinisikan pada W3C (*World Wide Web Consortium*) dan IETF (*Internet Engineering Task Force*). WebRTC dipublikasikan sebagai teknologi

open-source oleh Google pada Mei 2011, dan API-nya secara *native* dikembangkan dalam bahasa JavaScript (Dutton et al., 2012). Terdapat beberapa komponen dan konsep utama dalam protokol WebRTC, salah satunya ialah komponen jaringan atau koneksinya yang disebut RTCPeerConnection.

Komponen RTCPeerConnection dalam WebRTC merupakan sebuah antarmuka yang merepresentasikan sebuah koneksi antara suatu komputer dan *peer* lainnya dalam suatu jaringan *peer-to-peer* (Alvestrand, 2021a; Jennings, Hardie, & Westerlund, 2013; Perkins, Westerlund, & Ott, 2021). Dalam sebuah jaringan WebRTC dengan skema *full mesh*, suatu komputer pada sebuah jaringan WebRTC dengan *n peers* akan memiliki $(N-1)$ RTCPeerConnection dengan setiap komputer lainnya dalam jaringan. Terdapat pula skema-skema lain yang mengoptimisasi bentuk jaringan *peer-to-peer* ini dengan keuntungan dan kerugian tertentu.

Penggunaan WebRTC dapat digunakan untuk pertukaran media berupa *stream* yang dikirimkan terus-menerus, sehingga WebRTC menyediakan komponen MediaStream yang merepresentasikan sebuah *stream* atau arus multimedia berupa suara atau video (Alvestrand, 2021a; Sredojev et al., 2015). Alvestrand (2021b) pada dokumen protokol IETF: *WebRTC MediaStream Identification in the Session Description Protocol* menyampaikan beberapa detail terkait MediaStream pada WebRTC. Sebuah MediaStream dapat mengandung satu atau lebih MediaStreamTrack yang merupakan *track* audio atau video. MediaStreamTrack dapat ditambahkan pada RTCPeerConnection yang nantinya dapat diterima oleh ujung lain dari koneksi tersebut. MediaStream akan menggunakan protokol UDP secara bawaan.

Salah satu komponen lain dalam WebRTC yang signifikan ialah RTCDataChannel yang dijelaskan secara detail pada IETF: *WebRTC Data Channels* (Jesup, Loreto, & Tüxen, 2021). RTCDataChannel merupakan kanal data yang digunakan untuk mentransmisikan data apa saja dalam sebuah RTCPeerConnection. Secara teknis, sebuah koneksi dapat memiliki hingga 65534 RTCDataChannel. Berbeda dengan MediaStream, RTCDataChannel dapat digunakan sebagai kanal untuk membagikan pesan teks atau biner antarklien.

API WebRTC juga menyediakan dua jenis mode pengiriman. Salah satunya ialah mode pengiriman pesan berurutan dan *reliable*, yang konsep pengirimannya sama dengan data yang ditransmisikan dengan protokol TCP (*Transmission Control Protocol*). Potensi penggunaannya dapat digunakan untuk pengiriman pesan atau berkas. API ini

juga menyediakan pengiriman pesan yang tidak harus berurutan dan memperbolehkan kekurangan pesan yang ekuivalen dengan UDP (*User Datagram Protocol*). Potensi penggunaannya bisa untuk permainan, pengendalian perangkat jarak jauh, serta banyak lagi karena mengurangi biaya komputasi *overhead* untuk setiap transmisi datanya, sehingga mode ini bertransmisi dengan lebih cepat. Terakhir, API ini menyediakan pengiriman pesan *partial reliable* dengan protokol SCTP (*Stream Control Transmission Protocol*) yang dapat didefinisikan waktu maksimal *timeout* dan maksimal transmisi ulangnya, urutan dari pesan juga dapat dikonfigurasi.

Sebelum memulai sebuah koneksi antar *peer* dan transmisi media dilakukan, suatu *peer* hendaknya mengetahui informasi *peer* lain yang terdapat dalam jaringan tersebut. *Signalling server* bertindak sebagai sebuah server yang mengelola koneksi antarperangkat, namun tidak mengelola lalu lintas media transmisi data itu sendiri (Petit-Huguenin, Nandakumar, Holmberg, Keränen, & Shpount, 2021). Server ini hanya sebagai perantara yang memberikan kondisi suatu jaringan dan menandakan *peer* mana saja yang masih terhubung dalam jaringan tersebut. Server akan bertanggungjawab untuk membolehkan sebuah *peer* untuk menemukan *peer* lain di dalam jaringan, mengarahkan pembuatan koneksi untuk *peer* baru yang masuk ke dalam sebuah jaringan WebRTC, serta mengulang, mematikan, atau melakukan *reset* sebuah koneksi bila diperlukan.

Proses *signalling* ini tidak didefinisikan caranya secara langsung dan memiliki banyak metode alternatif. Terdapat beberapa protokol yang bisa digunakan untuk melakukan *signalling*, antara lain XMPP (Extensible Messaging and Presence Protocol), XHR (XML HTTP Request), dan masih banyak lagi (Sredojev et al., 2015). Salah satu yang umum digunakan lainnya adalah SIP (Session Initiation Protocol) yang memanfaatkan koneksi WebSocket pada setiap klien dengan *signalling server* (Adeyeye, Makitla, & Fogwill, 2013). Proses *signalling* lebih lanjut didefinisikan menggunakan suatu protokol insiasi yang dikenal dengan SDP (*Session Description Protocol*).

SDP (*Session Description Protocol*) pada dasarnya berisikan informasi-informasi suatu peer kepada peer lainnya. Petit-Huguenin et al. (2021) menjelaskan secara detail prosedur inisiasi jaringan WebRTC yang menggunakan SDP ini pada dokumen IETF: *Session Description Protocol (SDP) Offer/Answer Procedures for Interactive Connectivity Establishment (ICE)*. Untuk memulai sebuah jaringan, terdapat sebuah objek informasi yang disebut Session Description Protokol yang akan ditawarkan kepada *peer* yang baru masuk ke dalam jaringan WebRTC dan berisi informasi-informasi tertentu menge-

nai *peer* yang menawarkan tersebut. Misalnya berupa alamat URL, jenis media yang ditransmisikan, *codec*, dan masih banyak lagi. SDP akan dikirimkan kepada *signalling server*. Setelah *peer* yang ditawarkan menerima, maka *peer* yang ditawarkan tersebut akan memberikan SDP-nya kepada *peer* yang menawarkan, sehingga sebuah jaringan WebRTC akan terjalin. Kandidat yang dapat menerima SDP ini dideskripsikan melalui sebuah ICE Candidate, yaitu sekumpulan rute yang dapat dilalui oleh sebuah *peer* untuk dapat meraih *peer* lain secara langsung. Di dalam SDP, terdapat deskripsi ICE Candidates ini. Dalam beberapa kasus, ICE Candidates akan dikirimkan melalui *signalling server* dengan metode *trickle*, yakni terpisah dari SDP dan ditambahkan satu per satu saat ada ICE Candidate baru yang didapat dari STUN server.

Dokumen IETF yang diajukan oleh Petit-Huguenin et al. (2021) juga membahas lebih lanjut mengenai ICE dan mekanisme pencarian alamatnya. Sistem alamat di Internet kebanyakan masih menggunakan protokol IPv4 yang secara praktis tidak dapat memenuhi semua kebutuhan penetapan alamat sehingga setiap perangkat memiliki alamat IP yang berbeda. Perangkat yang digunakan pada suatu jaringan dapat berada di belakang lapisan NAT (*Network Address Translation*). Mekanisme ini memetakan alamat IP privat menjadi alamat IP publik atau sebaliknya saat paket data bertransmisi dalam jaringan. NAT pada umumnya diimplementasikan pada sebuah jaringan dalam lingkup kecil, misalnya pada jaringan nirkabel rumah atau instansi tertentu. Pada WebRTC, untuk mengetahui alamat *peer* satu sama lain dibutuhkan suatu protokol yang disebut ICE (Interactive Connectivity Establishment). Server ICE akan mengembalikan ICE Candidate yang mendeskripsikan rute dan protokol yang harus diambil untuk mencapai suatu *peer* tertentu. Terdapat dua jenis server untuk ICE, yaitu STUN (*Session Traversal Utilities for NAT*) and TURN (*Traversal Using Relays around NAT*).

Server STUN merupakan server yang mengembalikan alamat IP publik terhadap *peer* yang menghubungi server itu sendiri, jenis NAT yang digunakan, dan *port* NAT yang diasosiasikan dengan *peer* tersebut. Pengembang dapat menggunakan server STUN publik nonkomersial, salah satunya milik Google. Apabila koneksi langsung antar-*peer* gagal dilakukan, maka TURN server berguna sebagai server perantara atau *relay server* yang meneruskan koneksi. Hal ini bisa terjadi karena adanya *firewall* yang diletakkan pada bagian mana saja dari jaringan yang memotong hubungan langsung lalu lintas dari WebRTC. TURN merupakan sebuah protokol untuk meneruskan lalu lintas jaringan yang tidak bisa dilakukan secara langsung tersebut. Sebuah TURN server memiliki alamat IP publik

yang dapat diakses oleh kedua *peer*, sehingga TURN Server ini dapat bertindak sebagai sebuah jembatan dalam transmisi media antara dua buah *peer* dalam sebuah jaringan WebRTC (Matthews, Rosenberg, & Mahy, 2010).

Berdasarkan paparan di atas, WebRTC merupakan suatu protokol kompleks yang penggunaannya fleksibel dan berpotensi dalam banyak kasus penggunaan. WebRTC menyediakan jaringan transmisi data secara *peer-to-peer* dengan latensi rendah. Dalam penelitian ini, WebRTC merupakan salah satu teknologi yang dimanfaatkan dalam pengembangan aplikasi. Salah satu komponen lain yang diperlukan sebagai dasar aplikasi ialah pengetahuan mengenai editor kode yang bersifat kolaboratif dan algoritma yang mewujudkannya.

2.3 Editor Kode Kolaboratif

Editor kode merupakan sebuah peralatan atau aplikasi yang digunakan oleh seorang *programmer* untuk mengembangkan kodenya. Fungsi-fungsi dasar editor kode yang membedakannya dengan editor biasa misalnya sorotan *syntax*, indentasi otomatis, dan penyocokan tanda kurung otomatis (IntelliJ, 2011; Kinder, 2013). Selain yang disebutkan, masih ada fungsi-fungsi lain yang tidak ada pada editor teks biasa. Dalam penelitian ini, semua operasi yang digunakan pada editor kode dapat direduksi secara tidak langsung menjadi operasi-operasi pada editor teks biasa (*plain text editor*). Pada *plain text editor*, setiap karakter pada teks tidak mengandung informasi tambahan. Perhatikan ilustrasi pada gambar 2.1 yang menunjukkan perubahan *styling* yang dapat dilakukan pada *rich text editor*. Operasi semacam ilustrasi tersebut diasumsikan tidak dapat dilakukan pada editor kode karena setiap karakter dianggap tidak menyimpan informasi tambahan.

Gambar 2.1: Diagram Contoh Perubahan pada *Rich Text Editor*

Pada editor kode atau teks yang bersifat kolaboratif setiap pengguna memiliki replika dari suatu dokumen teks yang akan berakhir sama (C. Sun & Ellis, 1998; D. Sun, Xia, Sun, & Chen, 2004). Setiap pengguna bebas melakukan penyuntingan secara bersamaan tanpa ada larangan tertentu. Operasi lokal kemudian akan diterapkan langsung pada replika lokalnya tanpa ada jeda (Attiya et al., 2016; Lv, Cui, & Li, 2015). Operasi yang dilakukan oleh seorang pengguna akan dipropagasi pada setiap pengguna lain secara langsung dengan latensi minimal, sehingga sifat kolaborasi waktu nyata dapat terwujud. Terdapat beberapa algoritma yang akan mewujudkan konsistensi *state* atau keadaan dokumen pada setiap replikanya. Setiap operasi yang dilakukan oleh setiap pengguna akan menghasilkan dokumen identik yang merupakan hasil penyatuan atau konvergensi yang memenuhi suntingan operasi-operasi tersebut (C. Sun & Ellis, 1998; C. Sun et al., 2019; D. Sun, Sun, Ng, & Cai, 2019b; D. Sun et al., 2004). Operasi yang dilakukan bersifat komutatif, yang berarti terlepas dari urutan diterapkannya operasi pada suatu dokumen, hasilnya akan tetap sama melalui algoritma yang mewujudkan konsistensi ini (C. Sun & Ellis, 1998).

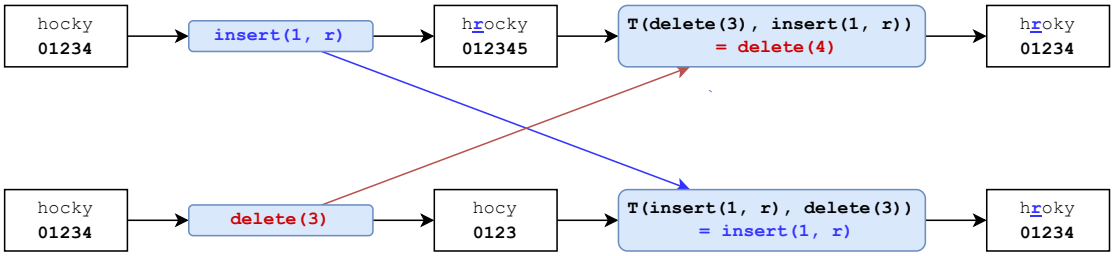
2.4 OT (*Operational Transformation*)

Salah satu tantangan dalam menciptakan suatu sistem terdistribusi adalah untuk memiliki suatu basis atau struktur data yang nilainya konsisten untuk setiap klien dalam sistem tersebut. Salah satu struktur data yang menjadi fokus penelitian adalah dokumen *plain text*. Metode OT (*Operational Transformation*) dikembangkan dengan motivasi bagi setiap pengguna dalam suatu sistem terdistribusi dapat memiliki dokumen yang sama untuk setiap perubahan yang terjadi (C. Sun & Ellis, 1998). Dalam algoritma dasar OT, operasi yang digunakan adalah `insert(pos, c)`. Operasi tersebut memasukkan sebuah karakter c pada indeks `pos` dan setiap karakter yang posisi awalnya berada $\geq$ `pos` akan digeser ke indeks selanjutnya. Ada pula operasi `delete(pos)` atau menghapus sebuah karakter pada indeks `pos` (Smith, 2012). Unit operasi seperti `insert` dan `delete` ini merupakan unit dasar dari OT (Smith, 2012).

OT dibuat untuk menyelesaikan konflik operasi yang dapat terjadi tanpa mengetahui urutan terjadi antara setiap kliennya. OT secara garis besar bekerja melalui sebuah fungsi T, yang mentransformasikan dan menyesuaikan parameter suatu operasi `op` yang akan dilakukan pada suatu dokumen, berdasarkan operasi-operasi sebelumnya yang telah diterapkan pada dokumen tersebut. Terdapat dua sifat yang umumnya harus dipenuhi oleh

suatu algoritma OT untuk bekerja, antara lain sebagai berikut (Li & Li, 2004; Martin, 2020; Ressel, Nitsche-Ruhland, & Gunzenhäuser, 1996; Smith, 2012).

- CP1/TP1 (*Convergent Property* 1 atau *Transformation Property* 1), yaitu $\mathsf{op}_1 \circ T(\mathsf{op}_2, \mathsf{op}_1) \equiv \mathsf{op}_2 \circ T(\mathsf{op}_1, \mathsf{op}_2)$.
- CP2/TP2 (*Convergent Property* 2 atau *Transformation Property* 2), yaitu $T(\mathsf{op}_3, \mathsf{op}_1 \circ T(\mathsf{op}_2, \mathsf{op}_1)) = T(\mathsf{op}_3, \mathsf{op}_2 \circ T(\mathsf{op}_1, \mathsf{op}_2))$.



Gambar 2.2: Diagram Ilustrasi TP1

TP1 berarti bahwa dokumen yang diterapkan op_1, dan dilanjutkan dengan penerapan operasi tranformasi op_2 terhadap op_1 haruslah ekuivalen dengan dokumen yang diterapkan op_2 dan dilanjutkan dengan penerapan operasi transformasi op_1 terhadap op_2. TP1 diperlukan hanya jika dua operasi yang hendak diterapkan pada suatu replika diterapkan pada urutan yang berbeda. Implementasi algoritma *operational transformation* yang memenuhi properti pertama ini cukup untuk membuat sebuah sistem terdistribusi dengan suatu sumber kebenaran yang berbasis *client-server*. Pada suatu sistem *operational transformation* yang tidak menggunakan TP2, salah satu syarat untuk mencapai konvergensi adalah setiap operasi hanya ditransformasikan pada satu sumber konteks yang sama (Xu & Sun, 2016). Misalnya riwayat setiap operasi-operasi yang terdapat pada server dianggap sumber operasi yang mutlak untuk setiap klien (Vidot, Cart, Ferrié, & Suleiman, 2000). Saat klien menerima pembaruan operasi, semua operasi lokal pada klien yang belum ditambahkan pada server akan ditransformasikan berdasarkan milik server. Operasi dari klien baru bisa di-*append* ke server jika semua operasi yang terdapat pada server sudah dimiliki klien.

Properti lainnya yang harus dimiliki oleh *operational transformation* ialah TP2 yang mengimplikasikan bahwa untuk tiga operasi berbeda, suatu operasi op_3 yang ditransformasikan terhadap dua operasi yang sudah diterapkan lainnya, yaitu op_1 dan op_2 dengan

sembarang urutan akan menghasilkan hasil transformasi yang sama. TP2 hanya dibutuhkan untuk sistem yang yang mengizinkan kedua operasi dijalankan pada dua *state* dokumen yang berbeda. Berbeda halnya dengan TP1, *update* secara *concurrent* dapat terjadi untuk klien yang berbeda. Karena kerumitan algoritmanya dan beberapa penelitian dibuktikan salah dalam mengimplementasikan OT yang memenuhi TP2 (Smith, 2012), CRDT dikenalkan sebagai alternatif dari OT sebagai metode untuk menjaga konsistensi dan kebenaran dari suatu dokumen atau data dalam sebuah jaringan sistem terdistribusi.

2.5 CRDT (*Conflict-Free Replicated Data Types*)

Sekitar awal tahun 2006, algoritma WOOT (*WithOut Operational Transformation*) yang diteliti oleh Oster, Urso, Molli, dan Imine (2005) dikenalkan sebagai sebuah algoritma non-OT pertama untuk memastikan sifat konvergen dari replika teks pada editor kolaboratif sebagai alternatif dari OT. Tidak hanya itu, konsistensi dari *intention* atau maksud dari sebuah operasi juga dapat dijaga (Li & Li, 2004). Misalnya ketika sebuah klien memasukkan sebuah karakter `X` di antara dua buah karakter `A` dan `B` pada sebuah *string* dengan penomoran indeks dari 0, "`ABCDE`". Operasi yang menjaga *intention*, memiliki makna bahwa masukkan karakter `X`, sehingga `A` mendahului `X` dan `X` mendahului `B`. Dengan kata lain, operasi `insert` yang direpresentasikan sebagai operasi $\texttt{insert}(1, \texttt{X})$ pada OT, direpresentasikan sebagai operasi $\texttt{insert}(\texttt{A} \prec \texttt{X} \prec \texttt{B})$ pada algoritma ini. Dengan `A` dan `B` merupakan suatu objek yang memiliki *id* pengenal yang bersifat unik untuk setiap karakter yang ada di dalam *string*. Algoritma ini menjadi awal dari perkembangan struktur data CRDT yang secara formal didefinisikan untuk tidak hanya pada *string*, namun berbagai struktur data abstrak lain pada tahun 2011 oleh Shapiro et al. (2011).

CRDT sendiri merupakan suatu tipe data abstrak untuk memelihara kecocokan dokumen pada beberapa replikanya dalam sebuah jaringan. Tipe data abstrak berarti semantiknya didefinisikan dari kumpulan nilai properti dan operasi fungsi atau prosedur. Oleh karena itu, implementasi dari CRDT untuk setiap operasinya bisa berbeda-beda, tapi menghasilkan *behavior* yang sama untuk operasi yang sudah didefinisikan. CRDT didesain untuk disimpan pada setiap klien atau *peer* dalam sebuah jaringan. Oleh karena itu, implementasi CRDT yang efisien terhadap memori dan waktu juga menjadi pertimbangan dalam menggunakan struktur data ini. Struktur data ini memiliki karakteristik pada setiap replikanya yang bisa dimodifikasi tanpa berkoordinasi dengan replika lain, bila setiap replika dilakukan operasi yang sama tanpa memerhatikan urutannya, maka semuanya akan

menghasilkan *state* atau keadaan akhir yang sama (Preguiça, Baquero, & Shapiro, 2018; Shapiro et al., 2011).

Salah satu dari contoh CRDT yang sederhana ialah *unordered set* atau himpunan tak berurut (Shapiro et al., 2011). Pada tipe data tersebut, setiap *peer* dapat melakukan operasi `insert(v)`, yaitu menambahkan suatu elemen `v` ke dalam *set* atau himpunan. Selanjutnya, ada pula operasi `erase(v)` yang akan menghapus elemen `v` dalam himpunan bila ada. Dalam penelitian ini, tipe data CRDT digunakan dalam proses pengolahan teks, sehingga operasi-operasi yang terkait dengan CRDT yakni serupa dengan yang disampaikan dengan operasi pada bagian 2.4, yakni `insert(pos, c)` serta `delete(pos)`.

Pada CRDT untuk teks biasa, setiap karakter akan memiliki *id* berbeda. Saat melakukan operasi `insert`, data perubahan akan disertai dengan dua *id* referensi karakter sebelum dan sesudahnya serta sebuah *logical clock* untuk menandai urutan pemasukan karakter, seperti ide serupa yang disampaikan pada bagian awal subbab ini (Oster et al., 2005). Terdapat beberapa cara yang umum diterapkan untuk merepresentasikan CRDT dan menangani kasus *operasi* `delete`. Cara pertama ialah dengan menyimpan karakter yang sudah dihapus pada suatu dokumen selamanya, dan hanya akan ditandai sebagai terhapus. Teknik ini dikenal dengan istilah *tombstone* (Molli, Urso, Imine, et al., 2006). Pendekatan yang lainnya ialah dengan memberikan *id* yang sudah terurut sesuai dengan urutan posisinya. Saat melakukan operasi `insert` pada suatu posisi tertentu, *id*-nya akan lebih besar daripada *id* karakter sebelumnya dan lebih kecil daripada *id* sesudahnya. *id* ini hendaknya bersifat dinamis dan ukurannya dapat bertambah seiring dengan bertambahnya karakter. Setiap *id* disertai dengan pengenal tambahan berbeda untuk setiap kliennya yang memastikan *id*-nya tidak ada yang duplikat. Saat melakukan operasi `delete`, *node* yang hendak dihapus tidak perlu disimpan pada suatu dokumen selamanya karena properti dari urutan *id* ini.

Bila dibandingkan dengan OT yang hanya memanfaatkan fungsi transformasi operasi dan dokumennya direpresentasikan secara minimal sebagai *array* karakter saja, CRDT membutuhkan struktur data tambahan untuk setiap karakter pada dokumen. Karakter tersebut akan diberikan *id* dan dikenalkan sebagai objek. Karakter tersebut juga akan memiliki urutan penghubung (implisit maupun eksplisit) seperti yang disampaikan pada dua pendekatan CRDT di atas yang menyatakan urutan karakter sebelum dan sesudahnya.

2.6 Perbandingan Lanjut CRDT dan OT

Industri teks editor kolaboratif waktu nyata masih didominasi oleh *operational transformation* meskipun CRDT memiliki kompleksitas dan efisiensi yang diklaim lebih optimal dibandingkan OT (D. Sun et al., 2019b). Tidak ada definisi perbedaan yang terlalu jelas untuk kapan suatu algoritma menggunakan OT dan CRDT, karena pada dasarnya keduanya merupakan transformasi untuk mencapai komutativitas dari fungsi yang direalisasikan menggunakan parameter yang berbeda. Pada CRDT, operasi diubah menjadi representasi *id* atau tanda pengenal lain, kemudian dikembalikan ke bentuk posisinya kembali setelah diaplikasikan, sementara pada OT, operasi dinyatakan dengan posisi dari karakternya langsung.

Pendekatan *operational transformation* berlangsung dan terfokus pada algoritma transformasi fungsi yang mengelola dan mengendalikan operasi dari setiap kliennya yang berlangsung bersamaan. Sementara CRDT terfokus pada konten seperti urutan objek, operasi yang berbasis *identifier* atau *id* pengenal, dan skema atau representasi lain dalam menyatakan operasi penyuntingan (C. Sun et al., 2019; D. Sun, Sun, Ng, & Cai, 2019a). Bila dilihat dari pembuktian dari kebenaran algoritmanya, CRDT memiliki kerumitan yang lebih tinggi dibandingkan OT karena adanya perlakuan khusus terhadap *state* kontennya (D. Sun et al., 2019a). Setiap CRDT membutuhkan kriteria khusus yang berbeda-beda untuk dapat memenuhi syarat konsistensinya. Hal ini bergantung dari struktur penyimpanan datanya. Struktur data CRDT yang spesifik untuk tipe data abstrak tertentu, misalnya untuk *set*, *map*, dan teks memiliki implementasi, algoritma, dan representasinya masing-masing. Variasi CRDT yang beragam ini menyebabkan pembuktian sifat konvergennya cenderung lebih sulit dibandingkan OT. Pada OT, syarat-syarat dasar tersebut sudah dibuktikan pada penelitian-penelitian pendahulunya dan variasi OT lain hanya perlu memenuhi syarat tersebut (Oster et al., 2005; Smith, 2012; D. Sun et al., 2004).

Perbedaan pendekatan ini menyebabkan kompleksitas dari kedua metode ini berbeda. Pada OT, variabel kompleksitas dipengaruhi oleh banyaknya operasi yang berlangsung secara bersamaan, dan tidak ada biaya untuk merepresentasikan dan memanipulasi karakter pada teks ke dalam bentuk objek dan struktur datanya. Pada CRDT, variabel kompleksitas ini akan semakin besar dan berbanding lurus dengan ukuran dokumen atau banyaknya konten. Selain itu, biaya *overhead* kompleksitas waktu dan memori inisialisasi untuk *peer* baru yang masuk ke dalam jaringan juga cenderung lebih besar dibandingkan OT.

Berdasarkan penelitian D. Sun et al. (2019a), kompleksitas waktu yang diperlukan untuk sistem OT yang merupakan *state-of-the-art* saat ini dalam mengaplikasikan operasi *remote* adalah $O(c)$ dan operasi lokal ialah $O(1)$, dan kompleksitas memori $O(c)$ atau $O(c \cdot m)$ dengan c adalah banyaknya banyaknya operasi bersamaan yang akan ditransformasikan. Dalam praktisnya nilainya sangat kecil $(c \leq 10)$ karena operasi berlangsung dalam waktu nyata. Terdapat juga variabel m, yang merupakan banyaknya pengguna dalam jaringan (umumnya $m \leq 5$).

Pada CRDT, definisikan C sebagai ukuran konten (tanpa *tombstone*) dan C_t sebagai konten seumur dokumen (dengan *tombstone*). Kompleksitas waktu untuk setiap operasinya dapat berkisar antara $O(C_t^2)$, $O(C_t)$, hingga $O(1)$ untuk CRDT berbasis *tombstone*, dan $O(\log C)$ untuk solusi yang tidak berbasis *tombstone*. Sementara kompleksitas optimal memorinya bisa mencapai $O(C)$ untuk solusi berbasis *non-tombstone* dan *tombstone* dengan *garbage-collection*. Nilai dari C umumnya berkisar antara $10^3 \leq C \leq 10^6$ untuk dokumen biasa pada umumnya.

2.7 Penelitian Terkait

Penelitian ini menggunakan *library* Yjs yang mengimplementasikan algoritma YATA (*Yet Another Transformation Approach*) untuk CRDT-nya. YATA menggunakan *linked-list* dalam merepresentasikan datanya dan menggunakan sistem *tombstone* dengan proses *garbage-collector* dalam mengoptimisasi objek konten yang dinyatakan terhapus. Dalam algoritma YATA yang diteliti oleh Nicolaescu, Jahns, Derntl, dan Klamma (2016), operasi `insert` dinyatakan sebagai menambahkan sebuah objek `o(id, left, right, isDeleted, content)` ke *linked-list*. Tanda pengenal `id` berisikan pasangan berurut *id peer* dan *logical timestamp* berupa operasi ke berapa yang telah dilakukan *peer* tersebut. Properti `left` dan `right` masing-masing merupakan variabel yang menandakan `id` untuk objek yang berada di sebelah kiri dan kanannya saat operasi `insert`. Bagi teks editor untuk mengetahui keberadaan suatu objek, terdapat properti `isDeleted` yang merupakan nilai *boolean* yang menunjukkan sudah terhapus atau tidaknya objek tersebut. Selain itu, terdapat properti `content` yang merupakan *string* yang berisi konten data dari objek.

Optimisasi pada Yjs terdapat pada properti `content` yang dapat berupa *string* atau kumpulan karakter. Saat ada operasi `insert` di antara dua konten dalam objek yang sama, objek tersebut akan dibagi menjadi dua objek lain. Sehingga dalam representasinya, informasi panjang konten juga akan disimpan pada setiap objek. Operasi penghapusan di-

lakukan dengan secara sederhana menandai objek tersebut menjadi terhapus. Untuk detail implementasi dan kompresi lebih lanjut disampaikan pada penelitian YATA (Nicolaescu et al., 2016).

Yjs merupakan *library* yang baru populer digunakan pada tiga tahun terakhir selama penelitian ini. Terdapat beberapa implementasi algoritma lain seperti Logoot yang menggunakan *vector clock* pada *timestamp* objeknya. *Vector clock* atau *state vector* secara sederhana merupakan sebuah vektor yang merepresentasikan versi dari setiap *peer* dalam jaringan. Dalam Logoot, operasi penghapusan dilakukan langsung dengan menghapus objek tanpa perlu menyimpan *tombstone* (Weiss, Urso, & Molli, 2009).

CRDT YATA melanjutkan ide awal dari CRDT LSeq (*Linear Sequences*) yang *id* pengenalnya tidak menggunakan *timestamp*. Permasalahan dari algoritma ini ialah dapat terjadinya *interleaving*, yakni urutan untuk operasi `insert` yang dilakukan bersamaan tidak memiliki komparator lanjutan (hanya *partial order*) saat adanya konkurensi yang terjadi ketika terdapat lebih dari satu karakter memiliki penunjuk objek `left` dan `right` sama pada dua *peer* berbeda (Kleppmann, Gomes, Mulligan, & Beresford, 2019; Nédelec, Molli, Mostefaoui, & Desmontils, 2013). YATA menyelesaikan masalah ini dengan menambahkan *logical timestamp* untuk setiap kliennya.

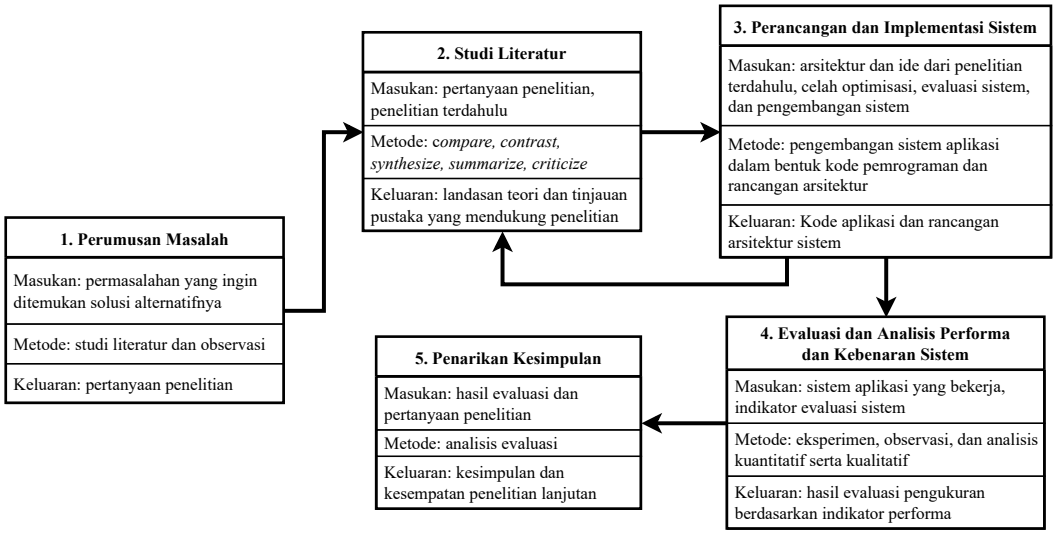
Alternatif dari YATA, yaitu RGA (*Replicated Growable Array*) menyelesaikan masalah *interleaving* pada LSeq dengan menggunakan pasangan berurut *id peer* dan *timestamp global* yang didapat dari *timestamp* selanjutnya dari *timestamp* operasi terkecil pada CRDT. Dalam praktisnya, algoritma ini memiliki berbagai optimisasi lebih terlepas dari algoritma yang disampaikan pada penelitian formalnya. Misalnya penambahan *garbage collector*, kompresi data antarjaringan, dan penyimpanan objek dalam memori yang memengaruhi performanya.

BAB 3
METODOLOGI PENELITIAN

Bab ini secara umum memaparkan tentang metodologi penelitian yang ditempuh dalam mengembangkan sistem PeerToCP yang mencakup pendekatan, perincian tahapan, serta aspek-aspek yang akan diujikan pada sistem. Pendekatan dan tahapan penelitian penting untuk memberikan penjelasan terhadap langkah-langkah saintifik yang ditempuh dalam penelitian ini.

3.1 Pendekatan dan Tahapan Penelitian

Penelitian ini dilaksanakan dengan pendekatan *experimental research*. Data kuantitatif dan kualitatif diukur untuk setiap variasi dari aplikasi PeerToCP yang memiliki implementasi *business logic* di atas UI (*user interface*) atau antarmuka pengguna yang sama. Variabel bebas dari penelitian ini difokuskan pada basis arsitektur dari jaringan PeerToCP, serta metode resolusi dan sinkronisasi data yang digunakan. Variabel terikat yang diukur dari penelitian ini disusun berdasarkan aspek-aspek sistem. Data kuantitatif didapat dari hasil *benchmarking* dan dianalisis secara deskriptif dan inferensial untuk mengetahui lebih jelas perbandingan performa antara setiap variasinya. Data kualitatif selanjutnya diperoleh melalui paparan deskriptif secara objektif terhadap sistem. Bagan berikut memberikan gambaran besar tahapan penelitian yang dilaksanakan.



Gambar 3.1: Bagan Alur Penelitian

Perumusan masalah dalam penelitian ini dilatarbelakangi oleh potensi penggunaan teknologi web berupa WebRTC dan beberapa variasi algoritma sinkronisasi data dalam suatu jaringan terdistribusi yang memiliki keuntungan dan kerugiannya masing-masing. Ide ini dikembangkan pula melalui kebutuhan sistem yang mempermudah melakukan kegiatan pemrograman kompetitif, yaitu suatu IDE (*Integrated Development Environment*) sederhana yang memungkinkan adanya pengembangan kode secara kolaboratif dalam waktu nyata dan penjalanan program yang dapat dilakukan pada suatu klien di dalam jaringan yang dapat diakses oleh setiap klien lain di dalam jaringan pula. Dari rumusan masalah tersebut, didapatkan pertanyaan-pertanyaan yang mendasari penelitian ini.

Melalui masukan pertanyaan, dilakukan studi literatur terhadap teknologi-teknologi dan penelitian terdahulu. Tahap ini menghasilkan landasan teori dan tinjauan pustaka sebagai dasar pengetahuan. Studi literatur dilakukan dengan membandingkan penelitian terkait yang serupa, dari segi performa, kerumitan implementasi, cara kerja, dan bukti kebenaran teknologi atau algoritma tertentu. Studi literatur ini berguna untuk mengetahui seberapa jauh kemajuan teknologi yang diteliti pada topik ini. Selanjutnya, penelitian dilanjutkan dengan perancangan dan implementasi sistem aplikasi PeerToCP. Terdapat tiga variasi dari sistem aplikasi PeerToCP yang diujikan, yaitu variasi dengan metode OT (*operational transformation*) berbasis *client-server*, CRDT (*conflict-free replicated data types*) berbasis *client-server*, dan CRDT berbasis *peer-to-peer*. Sistem yang telah dikembangkan kemudian dievaluasi secara objektif berdasarkan aspek-aspek tertentu yang merepresentasikan performa dan skalabilitas aplikasi. Poin-poin aspek yang disampaikan pada subbab 3.2 selanjutnya disampaikan untuk memberikan konteks perbandingan yang jelas antara suatu implementasi dengan yang lainnya.

3.2 Metode dan Skenario Evaluasi

Dari observasi terhadap beberapa aplikasi *real-time collaborative* lain, evaluasi pada penelitian ini mengikutsertakan beberapa aspek esensial untuk setiap variasi dari aplikasi PeerToCP, antara lain sebagai berikut.

1. *Correctness*, mengindikasikan kebenaran untuk setiap variasi implementasi PeerToCP. Aspek ini dipilih untuk menentukan bahwa setiap replika data yang dijaga kesamaannya berakhir konvergen dan identik.

2. *Lightweight*, aplikasi berjalan dengan sumber daya atau *resource* minimal dan tidak mengganggu jalannya aplikasi lain pada suatu sistem operasi. Suatu aplikasi hendaknya tidak menggunakan *resource* yang berlebihan dalam mencapai tujuannya, hal ini dapat berdampak langsung terhadap minat penggunaan aplikasi ke depannya.

3. *Responsiveness*, sinkronisasi replika data pada setiap klien dilakukan dalam latensi yang rendah dan layak guna. Setiap klien yang menggunakan suatu aplikasi *real-time collaborative* seharusnya mendapatkan jeda minimal untuk memberikan pengalaman pengguna atau *user experience* waktu nyata.

4. *Local-First*, operasi diterapkan pada replika lokal secara langsung setelah diberikan pengguna tanpa perlu berhubungan dengan klien atau server lain di dalam jaringan.

5. *Scalability*, aspek ini berhubungan langsung dengan setiap aspek lain, karena penggunaan arsitektur *peer-to-peer* pada awalnya memiliki motivasi untuk meningkatkan ketersediaan layanan dengan mengurangi beban pada server. Performa dari aplikasi berpengaruh terhadap banyaknya klien atau pengguna dalam suatu jaringan, sehingga aspek ini penting untuk diperhatikan dalam sistem terdistribusi aplikasi (Leibnitz et al., 2007).

Data kuantitatif dan kualitatif yang diperoleh pada proses *benchmarking* ini dianalisis. Setelahnya, hasil analisis disimpulkan dengan memberikan kesempatan optimisasi dan pengembangan untuk penelitian-penelitian selanjutnya.

BAB 4
DESAIN DAN IMPLEMENTASI

Bab ini menjelaskan detail arsitektur dan implementasi dari aplikasi PeerToCP. Selain itu, terdapat penjelasan singkat serta alasan pemilihan beberapa teknologi, termasuk *library* dan modul yang digunakan dalam implementasi, diikuti dengan *usecase* atau fitur sistem. Bab ini juga memaparkan detail skenario pengujian dan metode evaluasi disusun berdasarkan aspek-aspek sistem pada variasi yang akan dibandingkan.

4.1 *Library* dan *Framework* Terkait

Terdapat beberapa modul dan *library* pihak ketiga yang dimanfaatkan dalam pengembangan sistem aplikasi PeerToCP yang dibahas dalam penelitian ini. Berbagai *framework* dan *library* ini merupakan hasil penelitian dan implementasi oleh para pengembang sebelumnya. Pemilihan penggunaan setiap teknologi ini dipertimbangkan dengan alasan tertentu, salah satunya adalah Electron, yang menjadi basis pengembangan aplikasi *desktop* pada PeerToCP.

4.1.1 *Framework* Utama dan Komponen Editor Kode

Electron merupakan salah satu *framework desktop* yang sistem kerja pengembangannya serupa dengan komponen web. Bagian tampilan atau *frontend* dari Electron memanfaatkan aplikasi *Chromium* yang dapat mengolah bahasa *markup* web, seperti HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*), serta JavaScript. Bagian belakang atau *backend* dari Electron berjalan dengan *environment runtime* Node.js (Kredpattanakul & Limpiyakorn, 2018; Miglani & Shriram, n.d.). Node.js merupakan bahasa yang menggunakan *syntax* yang serupa dengan JavaScript dan dapat dikompilasi melalui kompilator yang disebut V8 engine (Tilkov & Vinoski, 2010).

Salah satu keuntungan menggunakan Electron adalah aplikasinya yang bersifat *cross-platform* atau dapat berjalan di beragam sistem operasi, seperti Windows, GNU/Linux, atau MacOS. Electron mengizinkan akses berbagai macam fungsi antarmuka sistem operasi, seperti memanggil subproses pada sistem dan menulis berkas. Karena *frontend*-nya yang menggunakan bahasa web pula, aplikasi yang dibuat dengan Electron cenderung

lebih mudah untuk dipindahkan dan diadaptasi dengan fungsi terbatas pada web. Selain Electron, masih terdapat beberapa alternatif *desktop-based framework* lain seperti Qt yang berbasis C++ dan Tauri yang berbasis Rust. Pemilihan Electron dipilih karena beberapa *library operational transformation*, CRDT, WebSocket, dan WebRTC yang umum digunakan sudah tersedia implementasinya dalam JavaScript dan dapat digunakan melalui Node.js.

Aplikasi dengan *framework* Electron memiliki proses utama atau dikenal dengan istilah *main process* yang akan berjalan saat aplikasi dimulai (OpenJS Foundation, 2022). Setiap jendela tampilan yang dibuka akan diwakili oleh sebuah proses pengolah atau *renderer process*. Tampilan PeerToCP direpresentasikan oleh sebuah jendela editor kode dengan *renderer process*-nya yang memiliki mekanisme untuk sewaktu-waktu berkomunikasi dengan *main process* dan membuat jendela *shell* yang mengandung program berjalan hasil kompilasi. *Renderer process* bertanggung jawab untuk mengolah konten *frontend* yang serupa dengan komponen-komponen web. Dalam penelitian ini, salah satu komponen utama yang ditampilkan pada jendela utama PeerToCP ialah CodeMirror.

CodeMirror digunakan dalam komponen editor kode dan berperan sebagai media interaksi pengguna dengan sistem aplikasi pada *frontend*. CodeMirror sendiri disusun untuk dapat di-*render* oleh peramban web dan mengolah teks dengan sistem *state* disertai operasi perubahan yang bersifat modular. CodeMirror menyediakan banyak ekstensi, aksesibilitas tinggi, serta dukungan untuk berbagai macam bahasa pemrograman. CodeMirror dimanfaatkan pula sebagai perantara dengan Yjs, sebuah *library* CRDT yang sudah diintegrasikan dengan CodeMirror.

4.1.2 Komponen Kolaborasi

Yjs merupakan sebuah *library* yang mengimplementasi CRDT yang disebut YATA (*Yet Another Transformation Approach*) (Nicolaescu et al., 2016). Yjs terdiri dari beberapa bagian, yaitu YDocs yang merupakan bagian utama implementasi berbagai struktur data untuk CRDT, ada pula YText yang merupakan variasi CRDT untuk operasi-operasi pada teks, serta YMap dan YArray yang dikombinasikan untuk menyimpan *shell* dan riwayatnya. Dalam penelitian ini, digunakan tiga struktur data CRDT tersebut yang abstraksinya berbeda-beda.

Yjs juga merupakan *library* yang tidak terpaku pada sebuah arsitektur. Dalam penelitian ini digunakan dua *provider* jaringan yang dapat berintegrasi dengan YDocs, yaitu

YWebRTC dan YWebSocket. Kedua provider ini masing-masing mengintegrasikan YDocs dengan jaringan berarsitektur *full-mesh peer-to-peer* serta *client-server* secara berturut-turut. Yjs memiliki banyak pengembang aktif dan hingga kini masih di-*maintain* dan dikembangkan, sehingga *library* Yjs dipilih dalam penelitian ini.

Untuk variasi *operational transformation* dari PeerToCP, penelitian ini memanfaatkan ekstensi *collaborative editing* dari CodeMirror yaitu @codemirror/collab. *Provider* jaringan untuk arsitektur *client-server* yang dikembangkan untuk metode ini menggunakan *library* rpc-websockets yang dimodifikasi sehingga dapat dimanfaatkan untuk melakukan *broadcast*, *specific-messaging* ke klien tertentu, serta fungsionalitas pemanggilan RPC (*Remote-Procedure Call*) berbentuk *promise* secara *asynchronous* dan *non-blocking*.

Pada variasi PeerToCP dengan metode OT, penyimpanan *shell* diimplementasi menggunakan prinsip OT dan sifat *append-only array* untuk menyederhanakan kompleksitas program. Operasi-operasi yang dapat dilakukan pada *shell* ialah berupa *keystroke* masukan pengguna. Secara khusus, beberapa *keystroke* diperlakukan oleh terminal secara harfiah dan program yang berjalan pada *shell* tersebut sendiri yang harus menangani bagaimana *keystroke* tersebut akan diperlakukan. Misalnya, operasi penghapusan atau melakukan *backspace* secara bawaan pada masukan *shell* diartikan sebagai menambahkan tiga karakter "\b \b" (tanpa tanda petik) atau ekuivalen dengan memindahkan *cursor* ke kiri, memasukkan karakter *whitespace*, dan memindahkan *cursor* ke kiri kembali. Dalam penelitian ini, aspek kolaborasi dari *shell* memperlakukan setiap klien untuk memberikan *keystroke* pada satu kanal yang sama tanpa *positional cursor* ganda seperti pada editor kode.

4.1.3 Komponen *Shell*

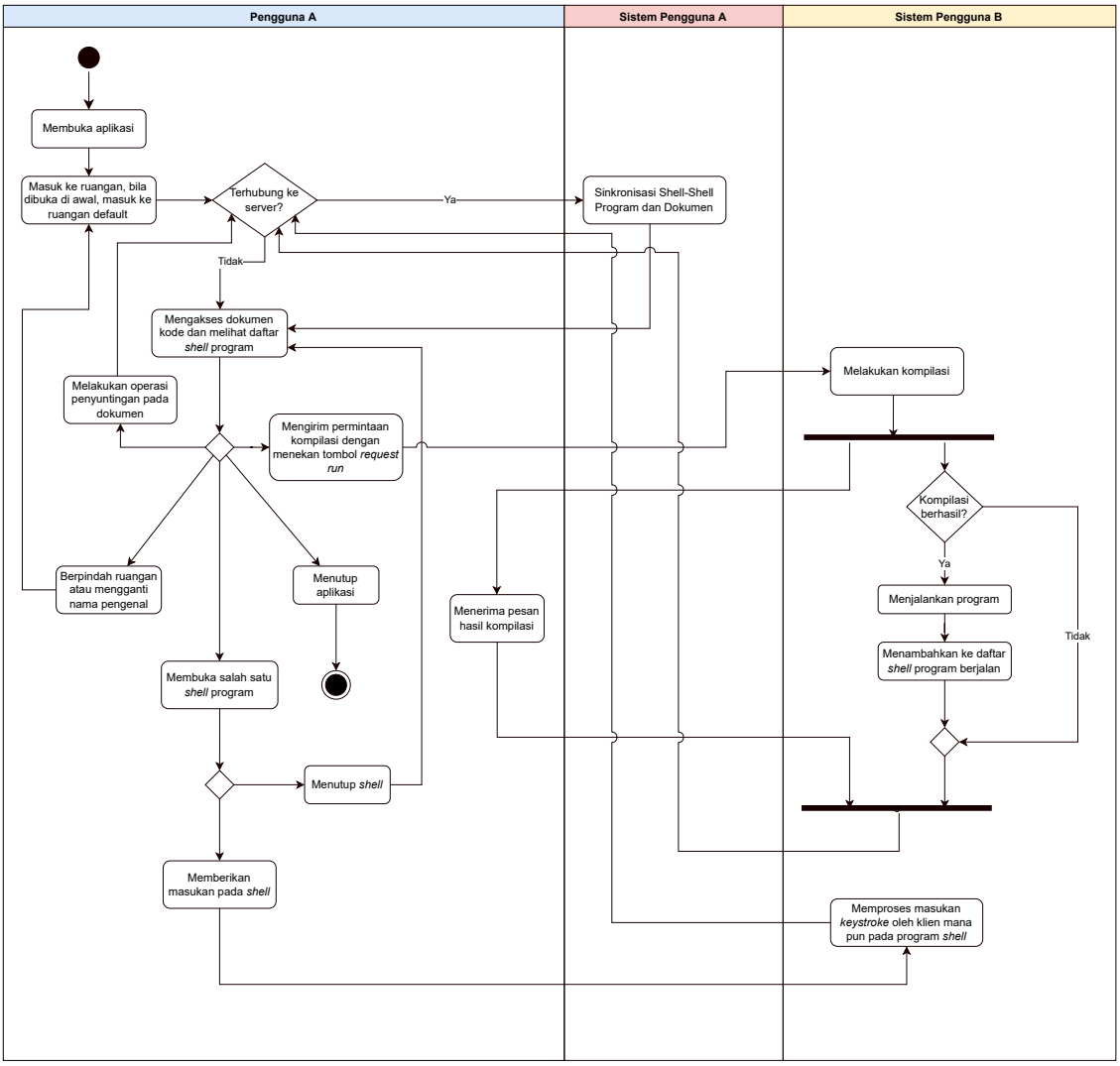
Untuk memenuhi komponen penjalanan program pada suatu *shell* yang dapat diakses oleh setiap klien dalam jaringan, dibutuhkan suatu *library* untuk mengakses sistem operasi untuk melakukan kompilasi terhadap kode. Kompilasi merupakan proses mengonversi kode dari bahasa dengan level yang lebih tinggi dan dapat dimengerti oleh manusia menjadi kode biner yang dapat dimengerti oleh mesin (Aho, Sethi, & Ullman, 1985). Pada penelitian ini, selain editor kode yang bersifat kolaboratif, proses kompilasi berkas kode tunggal juga dapat dilakukan oleh setiap pengguna. Proses kompilasi ini membutuhkan kompilator yang terpasang pada suatu sistem operasi. Pada Node.js, salah satu *library* yang dapat

digunakan untuk mengaksesnya ialah Node-pty.

Node-pty merupakan *library* Node.js yang menyediakan antarmuka untuk melakukan *fork* proses dengan deskriptor berkas *pseudoterminal*. Node-pty mengizinkan adanya aliran data untuk baca dan tulis dengan proses berjalan pada kernel. Node-pty berguna untuk menjalankan berkas hasil kompilasi yang bersifat CLI (*Command Line Interface*) yang tidak memiliki tampilan grafik untuk pengguna. Node-pty dipilih karena banyak digunakan dan bersifat *cross-platform*. Aplikasi ini juga membutuhkan bagian *frontend* untuk menampilkannya, dan digunakan Xterm.js. *Library* ini merupakan salah satu komponen yang menampilkan terminal pada web. Xterm.js memiliki antarmuka yang bisa menerima dan meneruskan data dari peramban (*browser*) yang dapat dihubungkan dengan sebuah proses berjalan pada sistem. Xterm.js dikembangkan tanpa memerlukan dependensi, sehingga dipilih dalam pengembangan sistem ini.

4.2 Desain Sistem

Aplikasi PeerToCP didesain sebagai sebuah aplikasi *desktop-based*, yang berarti dijalankan secara langsung pada komputer. Pilihan ini dipertimbangkan untuk mempermudah akses secara luring, sehingga tidak dibutuhkan koneksi internet untuk mengakses aplikasi. Selain itu, fitur kompilasi pada suatu *peer* memerlukan akses *system-call* yang tidak disediakan pada API peramban web (Firefox, 2022; Google, 2022). Peramban web ditetapkan seperti ini untuk mencegah *script* yang berpotensi menyerang sistem operasi komputer tidak dapat dijalankan secara langsung. *Activity diagram* yang menunjukkan garis besar alur penggunaan aplikasi dapat dilihat pada gambar 4.1.



Gambar 4.1: *Activity Diagram* Alur Penggunaan Secara *High Level*

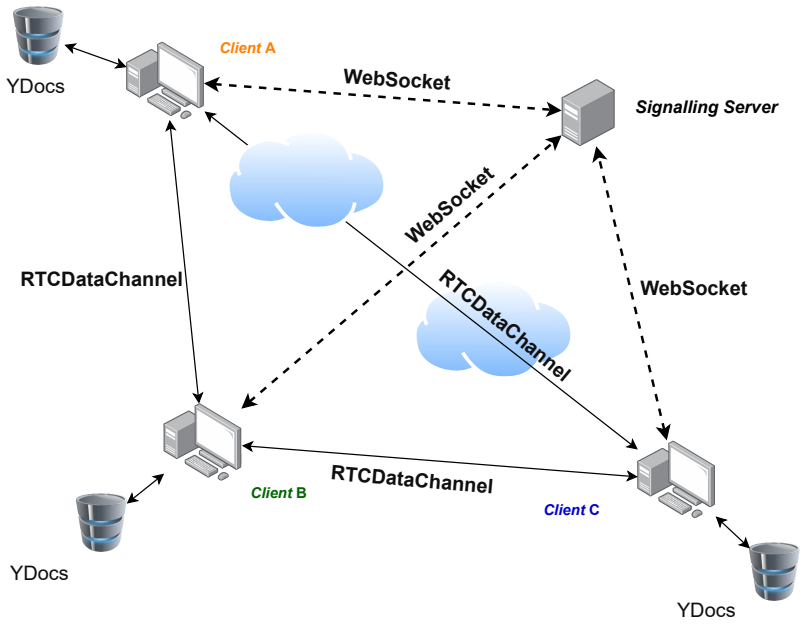
Saat pengguna membuka aplikasi, pengguna akan diarahkan untuk masuk ke ruangan awal secara *default*. Pengguna dapat melakukan operasi-operasi penyuntingan pada dokumen dan sinkronisasi dilakukan secara terus-menerus dengan *publish-subscribe design pattern* sehingga memberikan respons tanpa perlu mengecek atau *polling* saat terjadi *update* yang terjadi antarpengguna. *Request* atau permintaan kompilasi dapat diajukan kepada pengguna mana pun pada sebuah jaringan, termasuk permintaan kepada pengguna tersebut sendiri. Aplikasi PeerToCP akan mencoba mengirimkan pesan kepada pengguna yang ditentukan tanpa intervensi dari pengguna lain dalam jaringan. Apabila permintaan berhasil diterima, sistem pada penerima permintaan akan melakukan proses kompilasi dan memasukkan *shell* program berjalan dengan *id* tertentu ke daftar *shell* yang dapat diakses oleh setiap klien dalam jaringan. Daftar *shell* dan kontennya ini disimpan dalam

bentuk *object* pada *JavaScript*.

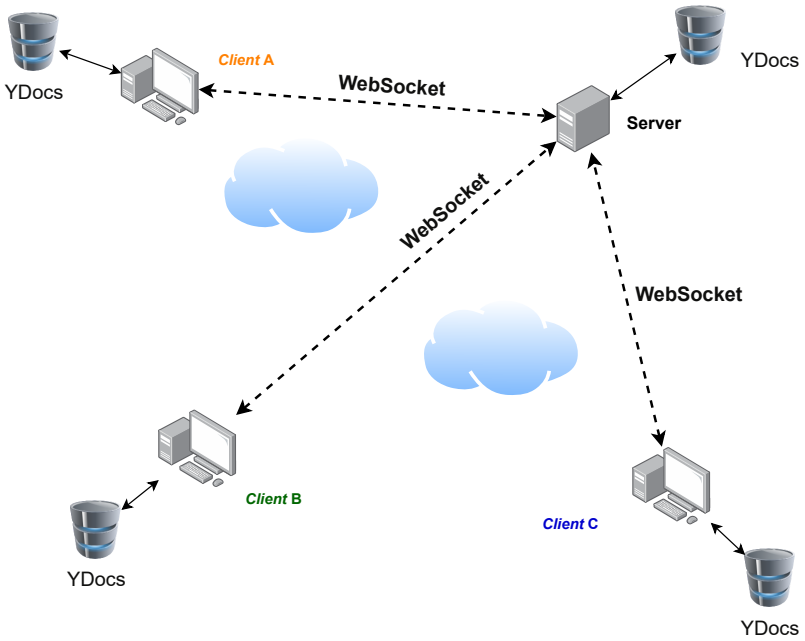
Pada gambar 4.1, perilaku sinkronisasi *shell-shell* pada program dan dokumen dilakukan bergantung dengan variasi implementasi dari program. Pada arsitektur *client-server*, sistem aplikasi pengguna A akan berhubungan dan melakukan sinkronisasi dengan server. Sementara pada arsitektur *peer-to-peer*, sistem aplikasi pengguna A akan berhubungan langsung dan melakukan sinkronisasi dengan *peer* lain. Selain itu, permintaan dan transmisi pesan hasil kompilasi dilakukan melalui pengiriman pesan secara langsung pada arsitektur *peer-to-peer*, namun harus melalui perantara server pada arsitektur *client-server*.

4.3 CRDT (*Conflict-Free Replicated Data Types*) Berbasis *Peer-To-Peer* dan *Client-Server*

Implementasi CRDT pada dua skenario arsitektur dalam penelitian ini menggunakan *library* CRDT Yjs. Setiap *peer* atau klien memiliki replika yang direpresentasikan dalam tipe data YDocs. YDocs dapat terdiri dari beberapa CRDT *nested* untuk menyimpan data-datanya, dalam penelitian ini *shell* direpresentasikan sebagai sebuah CRDT YMap yang memetakan *id* sebuah *shell* ke sebuah CRDT YArray yang berisi kontennya. Implementasi dari variasi *peer-to-peer* menggunakan *library* penyedia jaringan YWebRTC yang telah dimodifikasi dengan antarmuka untuk mengirim pesan ke *peer* tertentu melalui jaringan RTCDataChannel seperti yang ditunjukkan pada gambar ilustrasi 4.2. *Keystroke* yang diberikan oleh sebuah *peer* tertentu akan dikirim langsung pada *peer host* yang menjalankan programnya, kemudian *peer host* akan menambahkan hasil *keystroke* pada struktur *array* yang berkorespondensi dengan *shell* yang berisi program tersebut.



Gambar 4.2: Arsitektur *Peer-To-Peer* yang Menggunakan WebRTC dengan WebSocket *Signalling* Server dan CRDT



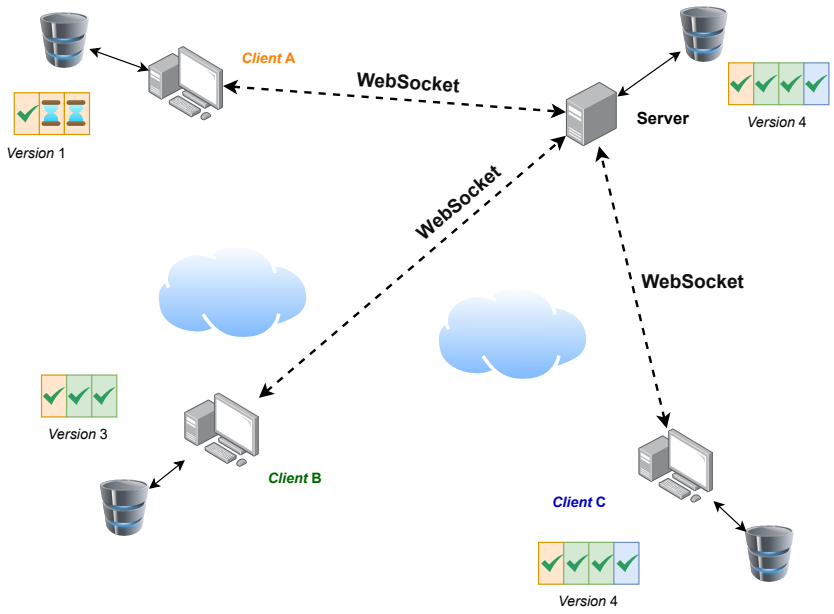
Gambar 4.3: Arsitektur *Client-Server* yang Menggunakan WebSocket dan *CRDT*

Perbedaan antara kedua implementasi arsitektur CRDT ini terdapat pada *provider* atau penyedia jaringannya. Arsitektur *client-server* diilustrasikan melalui gambar 4.3. Pada arsitektur *client-server*, penelitian ini menggunakan *library* YWebSocket yang telah di-modifikasi pula untuk dapat mengirim pesan kepada klien tertentu dalam sebuah jaringan melalui server. Penyedia jaringan memberikan abstraksi bagi setiap *peer* atau klien dalam

sebuah jaringan terdistribusi untuk berhubungan satu sama lain. Terdapat fitur tambahan bagi server pada arsitektur *client-server* untuk dapat melakukan persistensi data. Persistensi ini mengizinkan untuk server menyimpan salah satu replika dari dokumen, sehingga dokumen masih akan tetap ada walaupun tidak ada klien yang terhubung. Selain CRDT, arsitektur *client-server* digunakan bersama metode *operational transformation* untuk menjaga kesamaan replika dalam suatu jaringan sistem terdistribusi. Variasi PeerToCP dengan *operational transformation* yang hanya memenuhi *transformation property* 1 berbasis *client-server* menjadi salah satu variasi yang dibandingkan dalam penelitian ini.

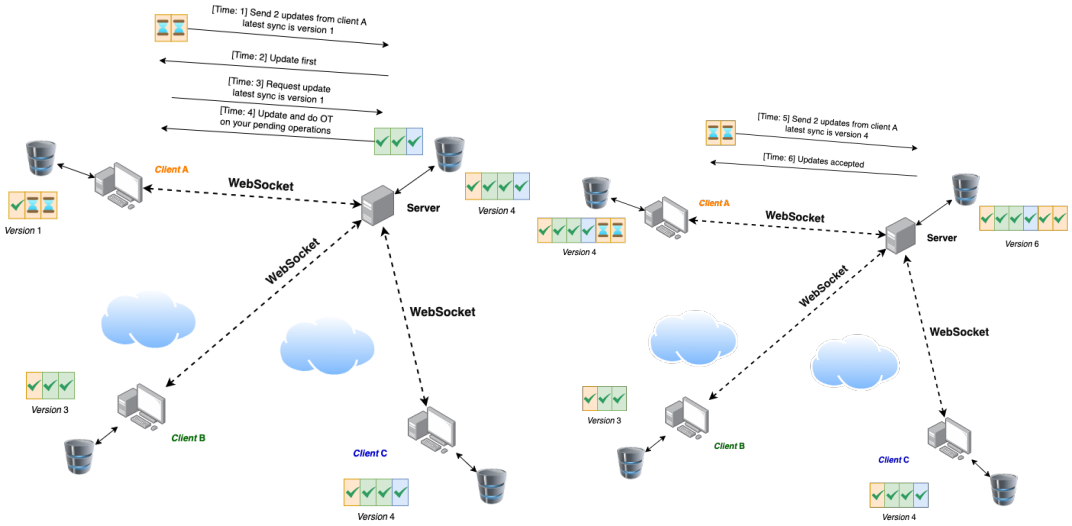
4.4 Metode *Operational Transformation* Berbasis *Client-Server*

Untuk suatu dokumen yang hanya menyimpan teks biasa, operasi *operational transformation* yang hanya memenuhi sifat TP 1 dapat diimplementasi dengan sederhana dalam arsitektur *client-server*. Aplikasi PeerToCP dengan variasi ini menggunakan *library* @codemirror/collab yang mengimplementasikan *operational transformation* yang menyimpan operasi perubahan CodeMirror dalam bentuk *delta*. Algoritma bekerja dengan menyimpan sebuah *array* perubahan lokal yang belum diketahui oleh server. Perubahan-perubahan lokal beserta versi dokumen terbaru dari server yang sudah diaplikasikan di lokal akan dikirimkan secara berkala dengan mekanisme RPC *over* WebSocket. Mekanisme RPC *over* WebSocket pada dasarnya mengizinkan alur *non-blocking* dalam menunggu balasan sebelum mengirimkan percobaan pengiriman perubahan lainnya. Apabila versi dasar dari server lebih baru daripada versi lokal, maka pengiriman perubahan akan ditolak, dan server akan memberikan respons kepada klien tersebut untuk melakukan *rebase* atau menambahkan *update* yang terdapat di server terlebih dahulu sebelum dapat mengirim percobaan pengiriman kembali.



Gambar 4.4: Arsitektur yang Menggunakan WebSocket dan *Operational Transformation*

Perubahan *remote* yang diterima dari server akan ditransformasikan satu sama lain dengan perubahan lokal yang belum diketahui oleh server. Perubahan lokal kemudian akan ditimpa dengan perubahan yang sudah ditransformasikan dengan perubahan *remote*. Percobaan pengiriman yang dikirim selanjutnya akan mengikuti perubahan yang sudah ditransformasikan ini. *Operational transformation* hanya terjadi di lokal dan perubahan akan diperbarui secara berurutan, sehingga setiap dokumen dalam jaringan memiliki replika yang berujung identik dan konvergen. Ilustrasi arsitektur ini ditunjukkan pada gambar 4.4.



Gambar 4.5: Ilustrasi Sinkronisasi pada Variasi *Operational Transformation*

Pembaharuan dan proses sinkronisasi PeerToCP dengan variasi *operational transformation* ini diilustrasikan pada gambar 4.5. Pada kasus ini, server tidak dapat menerima dua *pending update* milik klien A sebelum ia memiliki semua versi terbaru dari server. Setelah melakukan *pull update* dan menerima tiga operasi perubahan dari server, klien A akan mentransformasikan dua *pending update*-nya dengan suatu algoritma *operational transformation* terhadap tiga operasi baru ini. Kemudian klien A memperbaharui versi dokumen lokalnya menjadi versi 4, dan dapat mengirimkan *update*-nya. Server selanjutnya akan menambahkan dua operasi dari klien A tersebut serta menjadikan server memiliki versi 6.

Hal yang serupa diterapkan juga untuk *shell* program yang berjalan. Saat suatu klien memberikan *keystroke* pada salah satu *shell* yang aktif, *keystroke* akan dikirimkan melalui mekanisme yang serupa, namun akan diarahkan ke klien yang menjadi *host* atau menjalankan programnya. Selanjutnya, perubahan *shell* akan dikirimkan berkala pada server selama ada perubahan yang belum diterima atau diketahui server. Saat terjadi perubahan, server akan melakukan pengumuman kepada setiap klien untuk melakukan *pull*. Karena respons *shell* bergantung pada klien *host*, maka pembaharuan tidak perlu diterapkan fungsi transformasi seperti pada OT teks biasa. Salah satu aspek lainnya ialah *position mapping* yang tidak perlu ditangani karena kursor akan selalu berada di sebelah kanan karakter terakhir. Operasi seperti mekanisme *update* yang dilaksanakan secara serial ini diterapkan untuk mencegah terjadinya *race condition* pada saat gangguan jaringan.

Beragam arsitektur yang sudah dipaparkan pada subbab-subbab sebelumnya merupakan bagian *backend* dari suatu antarmuka editor yang sama, sehingga penilaian secara *end-to-end* yang akan dirasakan oleh pengguna menjadi tolok ukur yang logis dalam penelitian ini. Tolok ukur ini terdiri dari kasus yang mempertimbangkan aspek-aspek yang hendak diuji. Evaluasi dilakukan dengan perlakuan kontrol yang diusahakan identik untuk dapat memberikan hasil yang akurat.

4.5 Desain Evaluasi

Evaluasi sistem dilakukan dengan layanan Compute Engine Google Cloud Platform. Setiap variasi memiliki server yang di-*deploy* pada *instance* dengan zona daerah `asia-southeast1-b`, menggunakan mesin bertipe `e2-medium` yang memiliki 2 vCPUs dan memori RAM 4 GiB, serta sistem operasi Debian 11. Alasan dipilihnya mesin dengan tipe ini adalah untuk menghindari proses *benchmarking* atau tolok ukur yang dibatasi

oleh spesifikasi sistem. Dalam penelitian ini, *relay server* tidak disediakan untuk *peer* yang tidak dapat terhubung satu sama lain dalam sebuah jaringan *peer-to-peer* berbasis WebRTC. Penelitian ini mengasumsikan inisialisasi koneksi *peer-to-peer* WebRTC selalu berhasil pada variasi ini.

Untuk *instance virtual machine* yang mewakili pengguna, digunakan mesin bertipe `e2-small` yang memiliki 2 vCPUs, memori RAM 2 GiB, dan sistem operasi Debian 11. Mesin pengguna akan di-*deploy* pada *zone* `asia-southeast2-a`. Setiap mesin pengguna diletakkan pada *zone* yang sama, dan dekat dekat dengan server dengan tujuan untuk menyimulasikan perkiraan penggunaan aplikasi PeerToCP di dunia nyata. Terdapat empat skenario uji yang akan dilakukan terhadap sistem aplikasi dengan setiap skenario akan diujikan dengan $n \in \{2,4,8\}$ pengguna. Jumlah pengguna ini dipilih untuk menggambarkan skalabilitas pengguna dalam suatu jaringan. Selain itu, nilai maksimal n dipertimbangkan dengan perkiraan jumlah maksimal pengguna aplikasi PeerToCP yang tidak lebih dari delapan.

Skenario pertama menyimulasikan serangkaian operasi-operasi tertentu terhadap editor teks dengan jumlah *peer* sebanyak n yang berbeda-beda. Jumlah klien atau *peer* yang berbeda bertujuan untuk mewakilkan aspek *scalability*. Terdapat tiga macam operasi acak yang dilakukan sebanyak sepuluh operasi per detik dalam periode uji tiga menit, yaitu:

1. memasukkan (`insert`) suatu *string* karakter ASCII acak dengan panjang antara 2 hingga 5 secara inklusif;
2. menghapus (`delete`) suatu *range* dengan panjang antara 1 hingga 3 secara inklusif;
3. menimpa (`replace`) suatu *range* dengan panjang antara 2 hingga 5 dengan *string* karakter ASCII acak dengan jangkauan yang sama.

Operasi-operasi tersebut memiliki rata-rata penambahan sebanyak 15 karakter per detik dan perubahan sebanyak 41.67 karakter per detik (termasuk penghapusan) pada setiap kliennya. Tes dimulai bersamaan menggunakan *timer* bawaan sistem operasi yang secara *default* sudah disinkronisasi. Galat atau *error* yang terdapat pada sinkronisasi waktu *timer* tentunya tidak dapat dihindari, sehingga dianggap tidak ada. Namun, penelitian ini akan menggunakan *instance* yang sama untuk setiap *peer* atau kliennya sehingga mengurangi bias perbedaan *timer* pada setiap klien dalam jaringan. Selain itu, akan ada operasi pemutusan jaringan (*disconnect*) yang akan terjadi secara acak selama 30 detik di antara satu menit setelah uji skenario dimulai hingga satu menit sebelum uji skenario berakhir. Keadaan akhir editor teks setelah selesai hendaknya akan terhubung ke server

untuk melakukan sinkronisasi terakhir dengan *peer-peer* lain dari server. Selama editor teks tidak terhubung, operasi-operasi lain seperti `insert`, `delete`, dan `replace` masih berjalan pada editor guna menggambarkan aspek *local-first*.

Skenario pertama disusun selayaknya *stress-testing* terhadap operasi-operasi editor teks yang dapat terjadi di dunia nyata. Di akhir tes, sistem menunggu selama satu menit untuk memberikan kelonggaran sinkronisasi. Waktu terakhir *update* atau sinkronisasi akan disimpan untuk dibandingkan pada aspek *responsiveness* dan *scalability*. Nilai cacahan atau *hash* dari dokumen setelah uji skenario juga turut dicatat pada setiap klien atau *peer* serta dibandingkan satu sama lain untuk memenuhi aspek *correctness*. Selain itu, aktivitas RAM, CPU, dan transmisi data dalam jaringan pada setiap klien beserta server akan direkam menggunakan *framework* NetData untuk memenuhi aspek *lightweight* dan *scalability*.

Skenario kedua menggunakan *environment* yang sama dengan skenario sebelumnya. Skenario ini disusun untuk menguji *shell* pada setiap *n* klien berbeda, yaitu dengan menjalankan suatu program C++ yang akan mencetak 100 bilangan tak bertanda (*unsigned*) 64-bit acak dengan jeda acak setiap bilangannya selama 500 hingga 1500 milidetik. Selama itu, akan dilakukan operasi pemutusan jaringan (*disconnect*) yang akan terjadi secara acak selama 20 detik setelah 10 hingga 40 detik setelah program berjalan. Seperti skenario sebelumnya, waktu sinkronisasi terakhir, aktivitas perangkat, dan nilai cacahan dari setiap *shell* pada setiap klien akan dicatat dan dibandingkan untuk memastikan bahwa setiap pengguna memiliki replika *shell* yang sama.

Skenario ketiga dan terakhir berpusat untuk mengukur latensi dari editor teks dan *shell*. Pada skenario ketiga, *n* peer disusun serupa dengan dua skenario sebelumnya. Terdapat dua operasi yang akan dilakukan, yaitu memasukkan (`insert`) *timestamp* pada editor teks dan menimpa (`replace`) suatu *range* dengan panjang antara 10 hingga 15 karakter dengan *timestamp*. Setiap operasi yang dilakukan memiliki jeda acak selama 500 hingga 1000 milidetik guna menghindari operasi yang dilakukan secara bersamaan dalam periode yang tetap. Setiap klien atau *peer* kemudian menangkap *timestamp* dan mengukur latensi perbedaan saat *timestamp* dicetak dan *timestamp* diterima pada klien lainnya. Pada skenario keempat, setiap klien akan menjalankan program C++ yang akan mencetak 100 *timestamp* dengan jeda acak selama 500 hingga 1500 milidetik. Setiap klien kemudian menangkap setiap *event* ini dan mengukur latensinya dengan cara serupa, yaitu dengan menghitung selisih pemasukan konten *timestamp* oleh sebuah *shell host*

(pengguna yang menjalankan program) dengan penerimaan pada setiap pengguna lainnya. Pada skenario ketiga dan keempat ini, tidak ada operasi pemutusan pengguna dengan tujuan menghasilkan data yang akan dievaluasi pada aspek *responsiveness* yang lebih akurat. Secara ringkas, jenis skenario yang telah dipaparkan dapat dibedakan berdasarkan poin-poin pada tabel 4.1.

Tabel 4.1: Keterangan Skenario Pengujian

Skenario	**Mengukur latensi**	**Jenis evaluasi**	**Durasi (detik)**	**Dilakukan *disconnection***
1	Tidak	Editor kode	180	Ya
2	Tidak	*Shell*	100	Ya
3	Ya	Editor kode	120	Tidak
4	Ya	*Shell*	100	Tidak

BAB 5
HASIL DAN PEMBAHASAN

Bab ini membahas mengenai hasil dan analisis dari evaluasi yang telah dilaksanakan berdasarkan skenario-skenario yang telah disusun pada bab sebelumnya. Pembahasan analisis dibagikan berdasarkan aspek-aspek yang telah disusun untuk memberikan gambaran yang lebih jelas mengenai performa program dan penggunaan *resource* atau sumber daya yang dibutuhkan untuk mencapai performa tersebut. Selain itu, bab ini juga membahas pertimbangan variasi dari PeerToCP yang lebih baik digunakan dan faktor-faktor yang memengaruhi pertimbangan tersebut. Melalui pertimbangan tersebut, kelemahan dari sistem aplikasi turut disampaikan untuk perkembangan ke depannya.

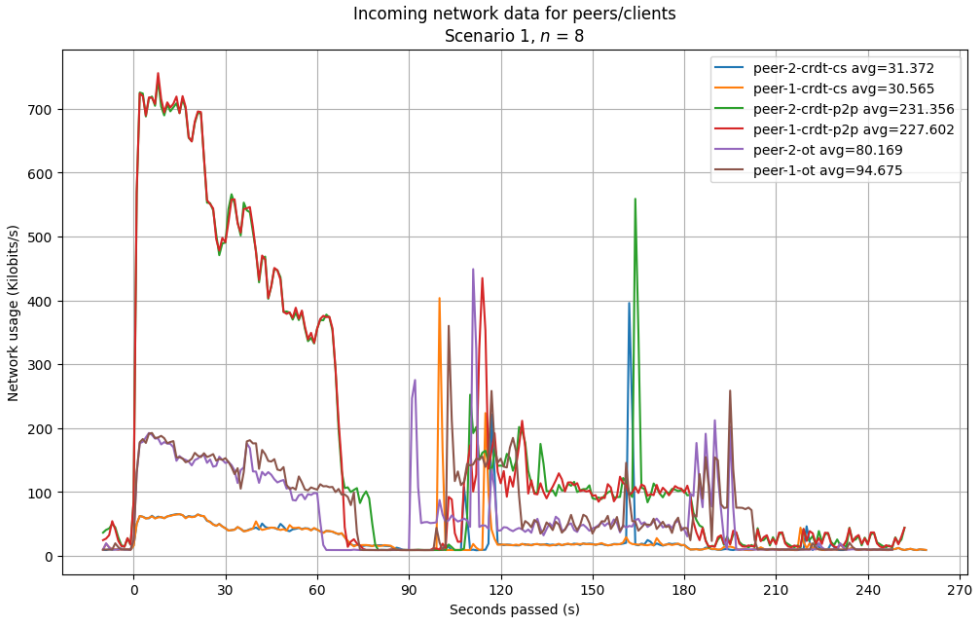
5.1 Aspek *Local-First* dan *Correctness*

Kedua aspek ini tidak dapat dipisahkan satu sama lain. Pada skenario pertama dan kedua, aspek *local-first* diuji dengan menyimulasikan proses pemutusan koneksi secara acak pada klien sementara pengubahan data terus terjadi pada setiap pengguna lokal secara luring. Berdasarkan eksperimen secara langsung, proses perubahan ini terjadi secara *local-first*, yang berarti perubahan lokal dapat terus dilakukan dan langsung diterapkan meskipun klien sedang tidak berada dalam jaringan. Ketika terjadi proses masuk kembali ke jaringan, data akan diperbaharui secara sesuai dengan perubahan yang dilakukan.

Berdasarkan eksperimen yang dilakukan, setiap variasi dari PeerToCP menghasilkan nilai hasil *hash* atau cacahan yang sama untuk setiap skenarionya, yang berarti setiap dokumen berada pada kondisi atau *state* akhir yang sama. Namun pada eksperimen yang dilakukan dalam skenario pertama dengan delapan klien pada variasi *operational transformation* berbasis *client-server*, terjadi pemutusan hubungan *disconnection* yang tidak dapat terhubung kembali. Diperkirakan hal ini terjadi karena ketidakefektifan metode transmisi data pada PeerToCP variasi ini dan menyebabkan kebutuhan *bandwidth* koneksi internet berada di luar batas *environment* pengujian.

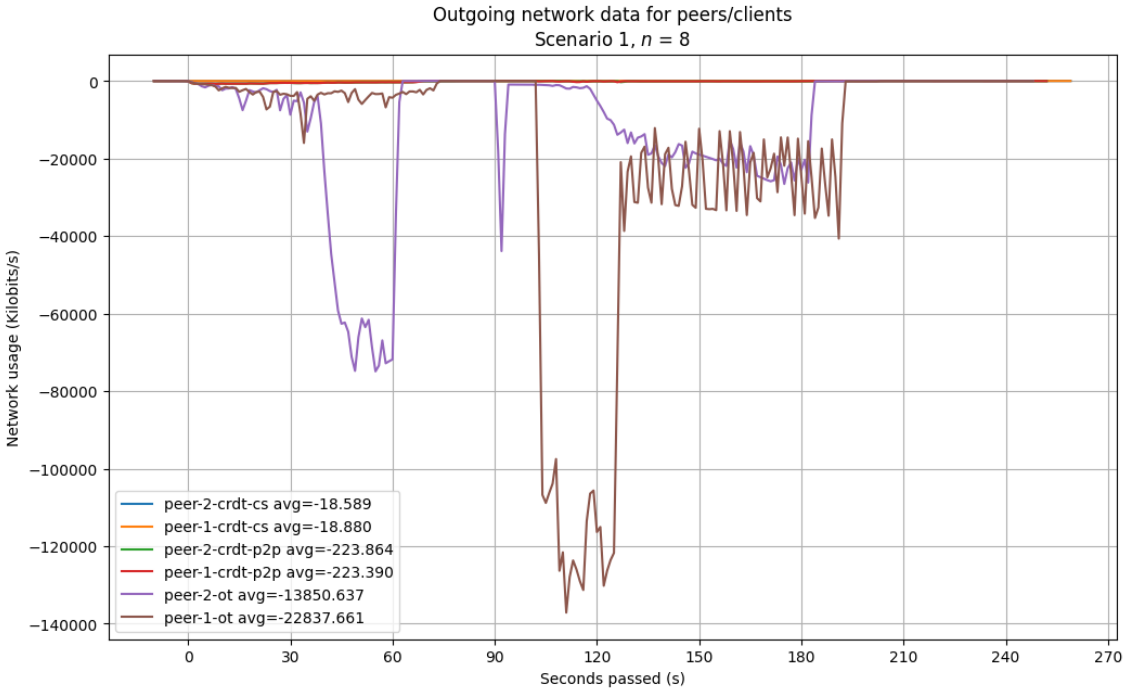
Eksperimen skenario pertama variasi OT *client-server* dengan jumlah klien delapan dilakukan kembali hingga tiga kali eksperimen. Dalam setiap eksperimen tersebut, satu hingga dua klien tidak dapat terhubung kembali. Pada eksperimen, waktu pemutusan

hubungan dilakukan secara acak dalam periode durasi yang sama untuk setiap kliennya seperti yang dijelaskan pada bab sebelumnya. Grafik lebih detail yang menyatakan transmisi data masuk melalui jaringan pada klien pertama dan klien kedua untuk setiap variasi aplikasi dalam skenario pertama ini ditunjukkan pada gambar grafik 5.1.



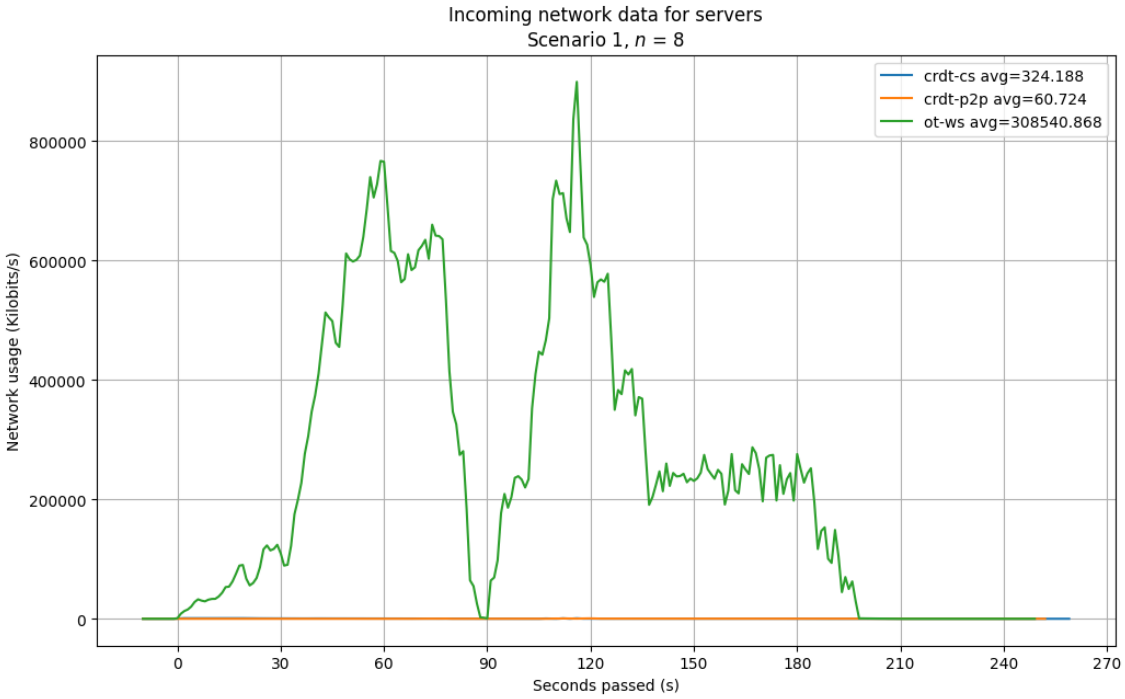
Gambar 5.1: Grafik Perbandingan Penerimaan Data pada Klien Pertama dan Klien Kedua untuk Skenario Pertama dengan $n = 8$

Grafik 5.1 menunjukkan aktivitas yang cenderung wajar karena ukuran *update* yang diterima tidak terlalu besar. Namun gambar grafik 5.2 menunjukkan klien pertama yang mengalami pemutusan jaringan pada detik ke-72 dan penghubungan kembali pada detik ke-102, terjadi pengiriman data yang cukup besar selama kurang lebih 20 detik. Begitu pula dengan klien kedua yang mengalami pemutusan jaringan pada sekitar detik ke-60 dan penghubungan kembali pada detik ke-90. Sesaat setelah suatu klien terhubung kembali ke suatu jaringan, terjadi *spike* pada jaringan yang cukup besar. Pada jumlah pengguna yang cukup banyak dalam satu dokumen, pemutusan dan penghubungan jaringan yang secara sengaja dilakukan pada waktu yang serupa dapat menyebabkan ketidakandalan pada sistem ini.



Gambar 5.2: Grafik Perbandingan Pengiriman Data pada Klien Pertama dan Klien Kedua untuk Skenario Pertama dengan $n = 8$

Pada saat operasi *update* dilakukan tepat setelah koneksi terhubung kembali, ukuran *update* cenderung lebih besar karena sudah terkumpul dari beberapa operasi yang terjadi selama klien sedang berada di luar jaringan. Ketika *update* ini dilakukan, server sedang dalam keadaan menerima *update* dari berbagai klien lain pula. Variasi algoritma *operational transformation* yang diimplementasikan dalam eksperimen ini hanya memperbolehkan *update* untuk dilakukan apabila versi terbaru di lokal milik klien sama dengan server. Apabila berbeda, maka server akan menolak *update* dan akan meminta klien untuk melakukan *pull update* terlebih dahulu. Sementara hal tersebut terjadi, tumpukan pengiriman *update* berukuran besar terus dikirimkan dan berpotensi memenuhi *bandwidth* server.



Gambar 5.3: Grafik Perbandingan Penerimaan Data pada Server PeerToCP

Lalu lintas jaringan pada server variasi *operational transformation* dengan arsitektur *client-server* dapat mencapai lebih dari 800000 *kilobits*/detik atau setara dengan 100 *megabytes*/detik. Melalui gambar 5.3, bisa terlihat bahwa rata-rata transmisinya sekitar 950 kali lebih besar dibandingkan variasi CRDT *client-server*. Perhatikan bahwa aktivitas pada server CRDT *peer-to-peer* yang menggunakan protokol WebRTC cenderung minim karena servernya tidak menangani transmisi pertukaran data, melainkan langsung berkomunikasi antar *peers*-nya.

Pada skenario pertama ini, variasi CRDT lebih dapat diandalkan karena paralelisasi *update* yang dilakukan terhadap dokumen. Faktor-faktor lain yang memengaruhi hal tersebut diperkirakan terdapat pada segi implementasi *provider websocket*, teknik kompresi, serta teknik *encoding* yang diterapkan. Selain itu, diperkirakan terjadi penumpukan *update* akibat operasi-operasi perubahan yang berukuran besar. *Update* tersebut lebih lambat penerimaannya dibandingkan *update* klien lain yang lebih sedikit. Hal ini menyebabkan beberapa klien harus melakukan *pull* berulang kali sebelum dapat meneruskan *update*-nya apabila versi lokal saat ini berbeda dengan yang terdapat pada *server*. Hal ini berakibat pada aspek lain pula, yaitu *scalability* dan *responsiveness*.

5.2 Aspek *Scalability* dan *Responsiveness*

Latensi dari teks editor secara khusus diukur secara terus-menerus pada skenario ketiga tanpa adanya pemutusan hubungan seperti pada skenario pertama dan kedua. Setiap pengguna diukur waktu selisih penerimaan datanya terhadap pengguna yang bukan dirinya sendiri dan dicatat latensinya pada waktu data dokumen diterima. Kemudian latensi tersebut dirata-rata untuk n pengguna pada jaringan. Tabel 5.1 merupakan statistik latensi untuk operasi nonlokal yang berarti latensi yang didapatkan dari pengukuran terhadap pengguna lain di dalam jaringan. Tabel tersebut menunjukkan aplikasi PeerToCP variasi CRDT *peer-to-peer* memiliki latensi yang paling rendah. Namun semakin besar nilai n, selisih perbedaan CRDT *client-server* dan CRDT *peer-to-peer* semakin mengecil. Sementara pada variasi OT *client-server*, latensinya dua kali lebih besar dari variasi CRDT *client-server*, dengan lonjakan latensi maksimum sekitar dua detik saat $n = 8$.

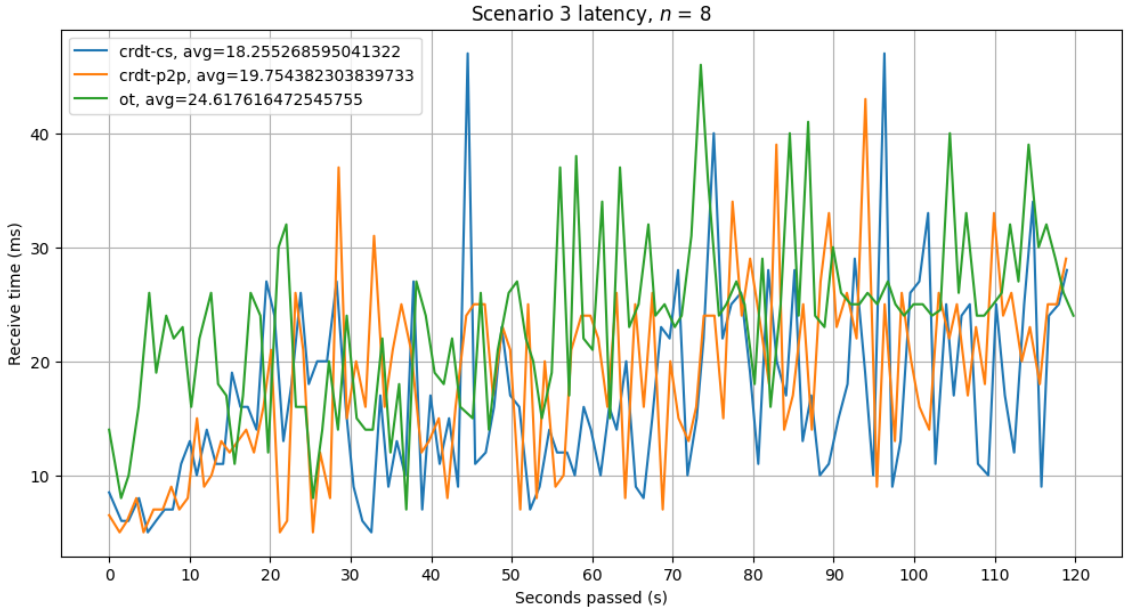
Tabel 5.1: Statistik Latensi Operasi Nonlokal pada Skenario Ketiga (Teks Editor Bersama) dalam ms

n	**CRDT *Peer-To-Peer***			**CRDT *Client-Server***			**OT *Client-Server***		
	Mean	Median	Max	Mean	Median	Max	Mean	Median	Max
2	23.30	21.00	76.00	39.50	38.00	134.00	87.42	84.00	168.00
4	34.00	30.00	223.00	46.10	42.00	220.00	98.84	86.00	1225.00
8	55.56	47.00	313.00	63.84	56.00	308.00	235.62	151.00	2010.00

Dari segi skalabilitasnya, dengan asumsi setiap pengguna berada pada jarak yang dekat, misalnya dalam eksperimen ini setiap *peers* berada dalam satu zona yang sama, yaitu Indonesia. Arsitektur *peer-to-peer* memberikan performa yang lebih optimal dibandingkan *client-server* untuk jumlah n tertentu. Berdasarkan perhitungan dan proyeksi jumlah pengguna yang bertambah, variasi dari CRDT *client-server* dapat memberikan *responsiveness* yang lebih baik untuk suatu nilai n yang lebih tinggi atau saat *peers* berada pada daerah yang lebih jauh antara satu dengan lainnya.

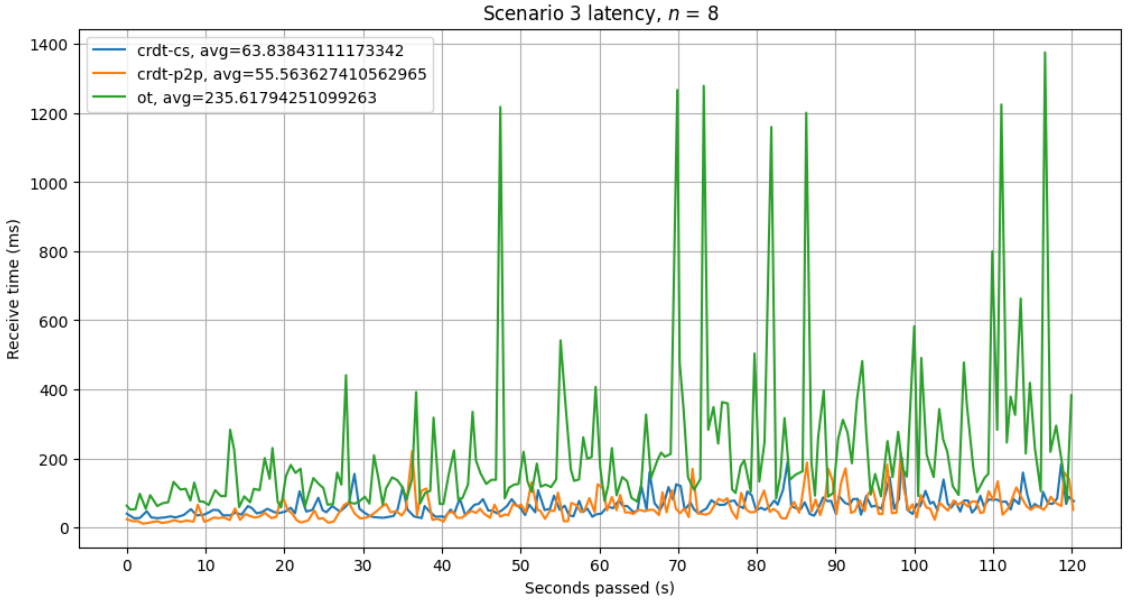
Perbedaan latensi ini dapat pula disebabkan algoritma yang memastikan bahwa dokumen tersebut sama pada setiap replikanya. Pada penelitian ini, operasi penerapan perubahan lokal diobservasi untuk mengetahui perbedaan latensi algoritma yang digunakan pada PeerToCP. Operasi lokal dalam konteks ini ialah hanya operasi yang dilakukan pengguna tersebut terhadap dokumen pengguna itu sendiri. Operasi ini bersifat *local-first* dan luring, yang berarti tidak melalui transmisi jaringan. Gambar 5.4 menunjukkan adanya

perbedaan latensi yang tidak terlalu signifikan, yaitu dengan rata-rata sekitar 5ms. Nilai lonjakan latensi maksimalnya juga cenderung tidak signifikan. Lonjakan ini dapat terjadi ketika operasi penambahan atau penghapusan karakter dilakukan pada indeks acak tertentu. Perlu diketahui bahwa skalabilitas latensi algoritma CRDT dan OT tumbuh secara berbeda. Kompleksitas waktu dari OT dipengaruhi oleh banyaknya operasi *update* berbeda, sementara kompleksitas waktu CRDT dipengaruhi oleh banyaknya karakter atau ukuran dokumen.

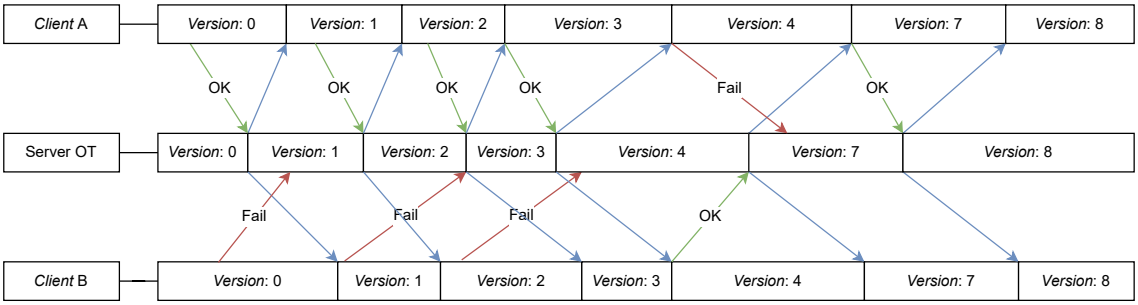


Gambar 5.4: Grafik Waktu *Resolve* Operasi Lokal pada Editor Aplikasi Pengguna PeerToCP

Latensi total diperoleh dari waktu penerapan operasi lokal dan nonlokal ditambah dengan waktu transmisi jaringan pada arsitektur yang berbeda-beda. Gambar 5.5 menunjukkan perbedaan latensi yang cukup besar, serta lonjakan latensi yang terjadi beberapa kali sepanjang skenario ketiga berjalan. Hal ini dapat disebabkan oleh beberapa hal, yakni waktu *resolve* operasi nonlokal dan adanya efek penumpukan *update* yang diilustrasikan pada diagram gambar 5.6.



Gambar 5.5: Grafik Rata-Rata Latensi pada Operasi Nonlokal Aplikasi Pengguna PeerToCP



Gambar 5.6: Diagram Ilustrasi Penumpukan *Update*

Pada arsitektur *operational transformation* berbasis *client-server*, *update* tidak akan diterima oleh server apabila suatu klien belum memiliki versi terbaru dari server. Penumpukan *update* ini terjadi ketika suatu klien tidak bisa mengirimkan datanya karena semakin besar dan selalu didahului oleh klien lain yang berada dalam jaringan tersebut. Hal ini menyebabkan terhambatnya pembaruan salah satu atau beberapa klien dalam jaringan dan mengakibatkan metrik pengiriman data pada klien dan penerimaan data pada server yang relatif jauh lebih besar dibandingkan variasi PeerToCP lainnya.

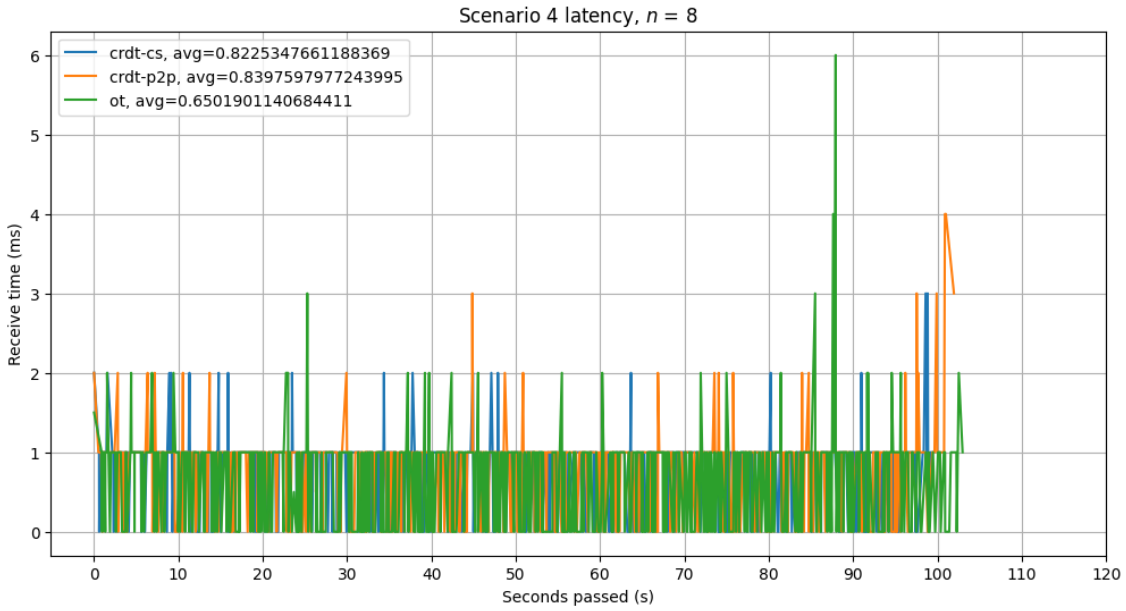
Dalam praktik penggunaan realistisnya, hal ini diperkirakan tidak umum terjadi karena frekuensi perubahan, pemasukan, dan penghapusan karakter tidak dilakukan secara bersamaan dan sebanyak skenario pertama dan kedua. Operasi salin dan tempel secara berkali-kali dapat berpotensi menyebabkan hal ini terjadi karena perubahan dokumen dalam jumlah karakter yang banyak. Namun, operasi tersebut diasumsikan hanya

dilakukan dengan jeda frekuensi yang membolehkan setiap klien dalam jaringan memperbaharui replikanya sebelum operasi besar lain dilakukan.

Selanjutnya, skenario empat menguji latensi *shell* yang dapat digunakan setiap pengguna dalam jaringan secara bersamaan. Secara umum, latensi tersebut dideskripsikan pada tabel 5.2. Berbeda halnya dengan *editor teks* yang waktu penerapan operasinya berdampak sebesar 20–70ms atau lebih. Struktur data berupa *dictionary* sederhana pada *shell* tidak memberikan perbedaan yang signifikan seperti yang digambarkan pada gambar grafik 5.7.

Tabel 5.2: Statistik Latensi Operasi Nonlokal pada Skenario Keempat (*Shell* Bersama) dalam Satuan ms

n	**CRDT *Peer-To-Peer***			**CRDT *Client-Server***			**OT *Client-Server***		
	Mean	Median	Max	Mean	Median	Max	Mean	Median	Max
2	3.05	3.00	5.00	16.75	17.00	19.00	31.79	30.00	170.00
4	3.13	3.00	6.00	16.55	17.00	21.00	44.47	32.00	309.00
8	3.36	3.00	19.00	17.16	17.00	35.00	59.86	30.33	551.00

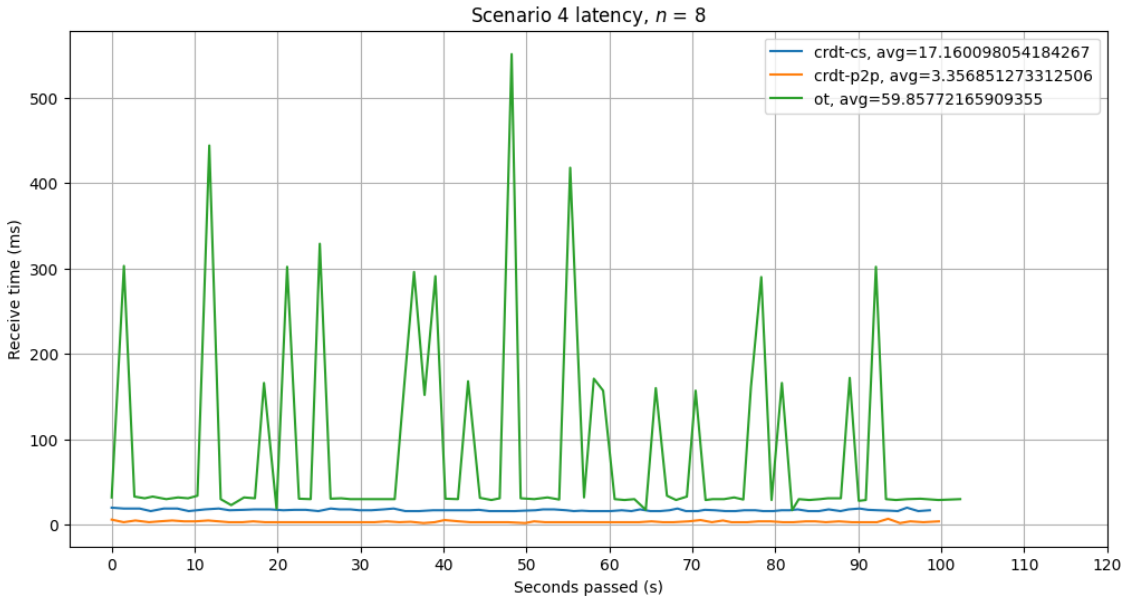


Gambar 5.7: Grafik Waktu *Resolve* Operasi Lokal pada *Shell* Aplikasi Pengguna PeerToCP

Hal ini dikarenakan operasi penambahan pada *append-only array* dalam *operational transformation* memiliki kompleksitas waktu konstan secara *amortized*. Dari aspek skalabilitas, operasi *shell* relatif hanya berpengaruh terhadap efektivitas dari transmisi data.

Update operasi *shell* mengalami permasalahan sama seperti pada skenario pertama, kedua, dan ketiga seperti yang ditunjukkan pada gambar grafik 5.8.

Faktor lain yang mempengaruhi alasan latensi *operational transformation* lebih lambat dari pada CRDT *client-server* ialah mekanisme komunikasinya. Implementasi CRDT Yjs dilengkapi dengan proses *encoding* dan *decoding* yang membuat transmisinya dalam jaringan untuk data yang besar dapat lebih cepat. Pada CRDT juga operasi *update* yang dilakukan secara berurutan tidak diperlukan. Sementara implementasi variasi *operational transformation* masih menggunakan mekanisme *update* yang sama dengan *editor teks*-nya.



Gambar 5.8: Grafik Rata-Rata Latensi pada Operasi Nonlokal Aplikasi Pengguna PeerToCP

Pada eksperimen ini, setiap *peers* pada arsitektur *peer-to-peer* dalam jaringan berada dalam jarak yang cenderung dekat karena berada dalam satu zona yang sama. Latensinya teruji lebih rendah dibandingkan kedua arsitektur *client-server* yang harus melewati server terlebih dahulu. Dalam penelitian ini, untuk $n \leq 8$, variasi *peer-to-peer* masih memberikan performa terbaik dari setiap variasi lainnya dari aspek *responsiveness*. Diperkirakan untuk suatu nilai $n > 8$, atau jarak antar *peers* yang lebih jauh, variasi CRDT dengan arsitektur *client-server* dapat dipertimbangkan sebagai alternatif yang lebih baik.

Dalam beberapa kasus lainnya, CRDT *peer-to-peer* dengan WebRTC dapat mengalami permasalahan antar-*peers* yang tidak dapat saling terhubung. Sehingga, alternatif berupa server penghubung, *relay* atau STUN server dapat digunakan dan latensinya memerlukan eksperimen lebih lanjut untuk dapat dibandingkan. Algoritma *operational*

transformation yang lebih baik juga dapat dipertimbangkan, karena secara teori operasi *update*-nya yang konstan dapat memberikan latensi lebih optimal karena memanfaatkan sifat *append-only array* dan fakta bahwa hanya satu host pengguna yang bertanggung jawab untuk meneruskan data interaksinya pada suatu *shell*. Untuk mendapatkan latensi yang lebih baik, struktur data CRDT dapat dimodifikasi lebih lanjut dengan memanfaatkan sifat tersebut.

Sistem dapat ditambahkan pula fitur untuk menentukan arsitektur optimal antara *peer-to-peer* tanpa *relay server* seperti pada eksperimen ini, *peer-to-peer* dengan *relay server* untuk menangani *peer* yang tidak dapat terhubung oleh WebRTC, atau pun *client-server* apabila banyaknya klien dalam suatu kelompok jaringan cukup banyak. Dari segi latensi, pertimbangan hal tersebut dapat ditentukan dengan metrik sederhana seperti *ping* antarpengguna yang dilakukan dari layar belakang aplikasi. Terdapat beberapa faktor lain yang harus dipertimbangkan pula, yakni aspek *lightweight* yang menggambarkan seberapa banyak proses, *bandwidth*, pekerjaan, serta memori yang harus disediakan oleh server dan penggunanya.

5.3 Aspek *Lightweight*

Aspek ini memaparkan seberapa besar *resource* komputasi yang dibutuhkan untuk setiap pengguna dan servernya. Semua metrik RAM pada penelitian ini diukur dengan melakukan normalisasi ke nol sesaat sebelum uji kasus dimulai. Normalisasi ini didasarkan untuk membandingkan perubahan pertumbuhan. Selain itu, dalam beberapa kasus dalam menjalankan klien, aplikasi elektron yang berjalan di atas Chromium memiliki sistem *garbage collection* dan *cache* (Thompson, 2015). Sehingga, pengukuran skenario diasumsikan lebih akurat bila disesuaikan relatif terhadap ukuran memori sesaat sebelum tes dimulai. Pada mulanya, setiap server mulai dengan RAM yang relatif rendah, yakni di bawah 20 MiB. Sementara setiap *peers* mulai dengan RAM yang cenderung lebih tinggi, yakni sekitar 200 MiB. Aplikasi ini membutuhkan memori yang cukup besar karena menggunakan *framework* Electron yang menjalankan peramban web Chromium pada *frontend*-nya.

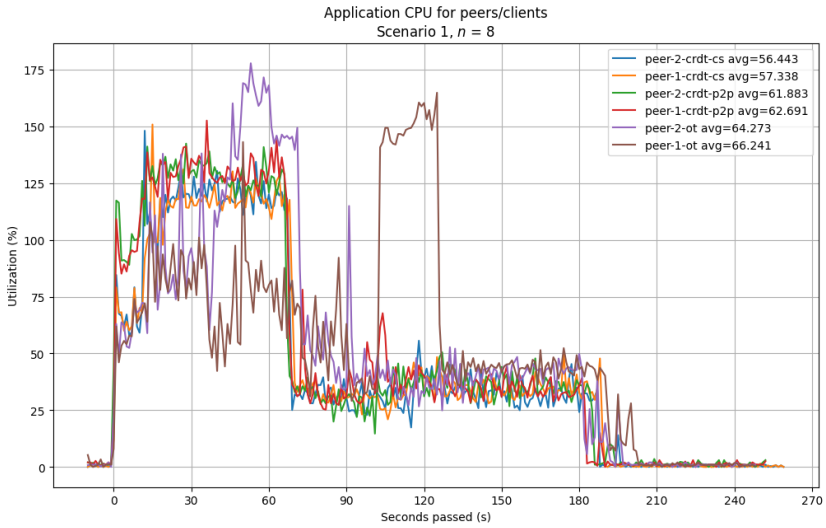
Tabel 5.3: Statistik Rata-Rata Aktivitas dan *Resource* Aplikasi Pengguna pada Skenario Pertama

n		**2**	**4**	**8**
CRDT *Peer-To-Peer*	CPU	39.1964893	60.13947488	62.2870496
	RAM	16.57856816	19.24442935	21.30499303
	Net In	28.5230253	96.16536701	229.4788353
	Net Out	-28.03809012	-93.97903876	-223.6266535
CRDT *Client-Server*	CPU	33.96390498	57.38616741	56.89049428
	RAM	14.92823532	19.12934751	20.94699478
	Net In	19.46582482	27.68880062	30.96869733
	Net Out	-19.76281471	-21.21326889	-18.73440885
OT *Client-Server*	CPU	47.79374975	73.29454876	65.25737338
	RAM	25.9460194	31.17265199	36.33549154
	Net In	62.24127487	99.81376692	87.42216719
	Net Out	-12962.04576	-28872.47276	-18344.14905

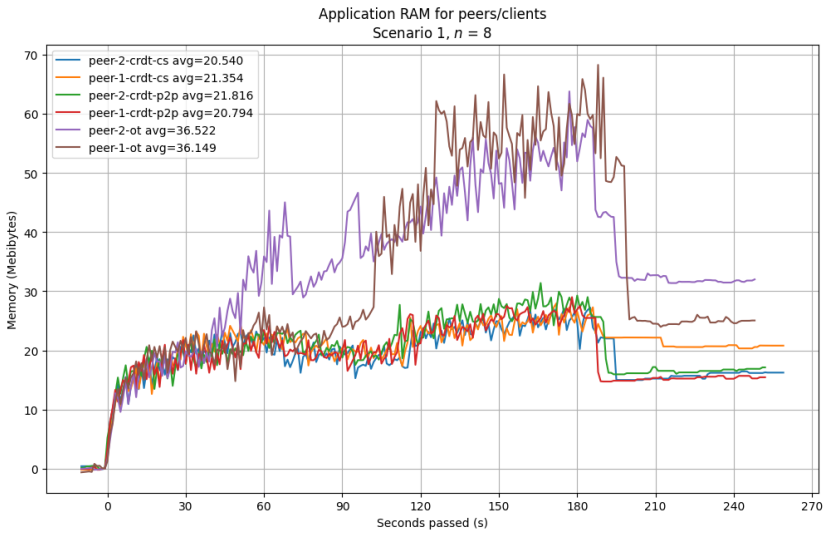
Tabel 5.3 memiliki parameter yang menunjukkan utilitasi CPU dalam satuan persen utilisasi. 100% utilisasi berarti sistem menggunakan satu *core* vCPU dengan penuh. Selanjutnya terdapat pula sumber daya atau *resource* RAM yang sudah dinormalisasi dalam satuan MiB, diikuti dengan kecepatan penerimaan dan pengiriman data dalam satuan *kilobits* per detik. Penggunaan RAM dan penerimaan data pada variasi CRDT *peer-to-peer* cenderung lebih tinggi untuk $n = 8$, hal ini disebabkan karena pada variasi ini, semakin banyak *peers* pada jaringan, berarti semakin banyak pula replika dokumen yang harus disinkronisasi. Karena *signalling server* pada variasi ini tidak berperan dalam transmisi data CRDT, sehingga setiap *peers* bertanggung jawab untuk saling memberikan informasi satu sama lain dalam memastikan kesamaan informasi pada setiap replikanya. Dari segi banyaknya data yang masuk ke dalam aplikasi, CRDT memastikan bahwa data-data tersebut dapat digunakan secara langsung dan setiap data yang dikirimkan akan digunakan tanpa ada kasus penolakan *update* seperti pada variasi OT. Sehingga dari segi transmisi data keluar, variasi CRDT dianggap lebih memenuhi aspek *lightweight* dibandingkan variasi OT.

Variasi aplikasi CRDT *client-server* secara relatif memberikan performa yang lebih baik dibandingkan dengan jenis lain. Dari segi utilisasi CPU dalam *environment* pengujian dengan dua core vCPU (utilisasi maksimal 200%), setiap variasi pada skenario

pertama dengan $n \geq 4$ secara rata-rata hanya menggunakan sekitar 1/3 *resource* dari CPU seperti yang terlihat pada gambar grafik 5.9. Metrik ini dapat dipertimbangkan dengan tambahan bahwa skenario ini memberikan muatan operasi-operasi yang relatif lebih banyak dari perkiraan aktivitas penggunaan normal. Selain itu, sistem operasi secara optimal akan menentukan proses yang harus diutamakan baik dari segi memori dan prioritas penggunaan CPU saat berjalannya aplikasi. Sehingga tidak menutup kemungkinan, penggunaan *resource* ini dapat lebih besar atau kecil tergantung dengan aktivitas lainnya yang terdapat pada suatu mesin.



Gambar 5.9: Grafik Perbandingan Utilisasi CPU pada Klien Pertama dan Klien Kedua untuk $n = 8$



Gambar 5.10: Grafik Perbandingan Penggunaan RAM pada Klien Pertama dan Klien Kedua untuk $n = 8$

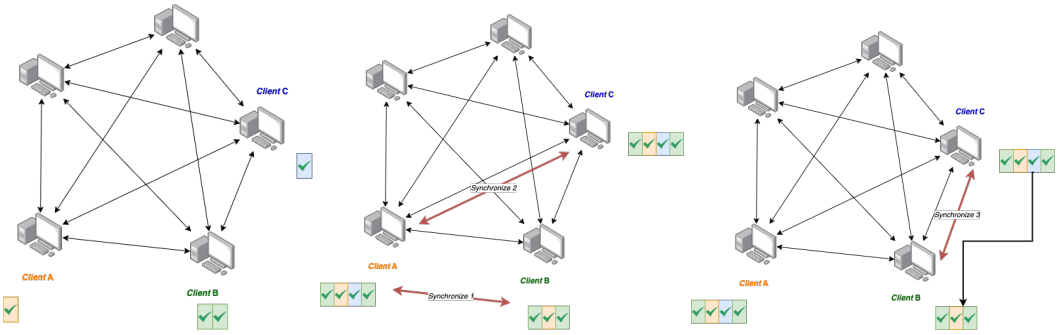
Penggunaan memori sebelum dinormalisasi rata-rata dimulai dengan menggunakan

sekitar 200 MiB, diikuti dengan pertumbuhan penggunaan seperti yang terlihat pada gambar grafik 5.10. Dari segi memori, kedua variasi CRDT memberikan pertumbuhan yang lebih optimal dibandingkan variasi OT. Namun, setiap variasinya menggunakan memori yang relatif kecil terhadap kapasitas penuh perangkat *environment* pengujian, yakni hanya sekitar 15%. Sistem operasi memiliki cara tersendiri dalam mengelola CPU dan RAM yang *idle*. Selama aplikasi tidak menggunakan utilisasi penuh yang mengganggu jalannya aktivitas pengguna dalam berinteraksi dengan komputer, maka setiap variasi Peer-ToCP dianggap memenuhi aspek *lightweight* dari segi utilisasi CPU dan RAM.

Tabel 5.4: Statistik Rata-Rata Aktivitas dan *Resource Server* pada Skenario Pertama

n		**2**	**4**	**8**
CRDT *Peer-To-Peer*	CPU	0.01	0.02	0.10
	RAM	0.67	-0.03	0.40
	Net In	10.72	14.81	60.72
	Net Out	-10.24	-15.79	-83.11
CRDT *Client-Server*	CPU	0.93	1.29	1.68
	RAM	5.01	6.83	9.54
	Net In	73.26	176.64	324.19
	Net Out	-71.79	-193.90	-391.66
OT *Client-Server*	CPU	4.85	14.18	24.33
	RAM	30.92	41.88	69.99
	Net In	52318.08	167768.04	308540.87
	Net Out	-26494.76	-84932.31	-156178.28

Aspek *lightweight* lainnya yang perlu dipertimbangkan ialah penggunaan *resource* atau sumber daya pada server yang dideskripsikan pada tabel 5.4. Dalam eksperimen ini, server *signalling* WebRTC pada CRDT *peer-to-peer* menggunakan *resource* RAM dan CPU yang minim. Berbeda halnya dengan arsitektur *client-server*, selain berperan dalam memelihara koneksi kelompok klien dalam suatu ruangan atau kelompok jaringan, server juga menjaga konsistensi replika pada setiap klien. Data yang dikirim dan diterima pada sebuah *peer* pada arsitektur *peer-to-peer* memiliki ukuran 1.5 hingga 2.5 kali lebih besar dibandingkan *bandwidth* data server pada CRDT *client-server*.

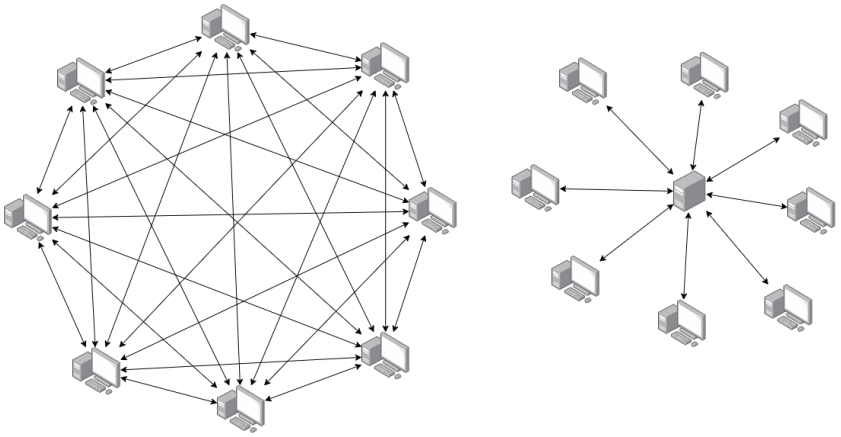


Gambar 5.11: Ilustrasi Sinkronisasi PeerToCP Variasi CRDT Berbasis *Peer-To-Peer*

Implementasi CRDT pada library Yjs menggunakan *logical timestamp* berupa *state vector* yang mencatat versi dari setiap pengguna dalam sebuah jaringan. Saat dua buah pengguna saling bersinkronisasi, hanya perbedaan antara dua buah versi yang didapat dari *state vector* tersebut yang akan dikirimkan. Dalam beberapa kasus, bila perubahan dokumen terlalu berbeda, seluruh konten dokumen secara langsung dapat dikirimkan untuk menghemat ukuran. Perubahan dokumen dari *state vector* ini bersifat komutatif dan idempoten, yang berarti perubahan yang sama dapat diterapkan berulang kali dan dengan urutan sembarang tanpa mengubah sifat konvergensi dokumen. Karena adanya sifat dari CRDT ini, pengiriman data kepada setiap klien pada CRDT tidak dipengaruhi banyaknya operasi yang dilakukan, melainkan cenderung dipengauruhi ukuran perubahan dokumen yang terjadi saat sinkronisasi dilakukan.

Pada gambar ilustrasi 5.11, apabila klien A melakukan sinkronisasi dengan klien B terlebih dahulu, kemudian diikuti dengan sinkronisasi klien A dengan klien C dan sinkronisasi klien B dengan klien C, maka klien C hanya perlu memberikan perbedaan yang belum didapatnya dari klien A. Secara khusus, apabila terdapat operasi penghapusan pada objek di struktur data yang belum disinkronisasi, konten objek tersebut tidak perlu dipropagasi. Kasus ini merupakan salah satu optimisasi Yjs yang dapat mengurangi total *bandwidth* yang perlu ditransmisikan antar pengguna pada variasi CRDT.

Dalam beberapa referensi implementasi, penyimpanan dokumen pada arsitektur *client-server* dapat ditambahkan juga dengan basis data eksternal untuk dapat menampung lebih banyak dokumen pada suatu kelompok jaringan. Dalam eksperimen ini, basis data pada setiap variasi tidak menggunakan teknologi pihak ketiga dan hanya menggunakan struktur data bawaan sederhana pada server. Sementara pada arsitektur *client-server*, persistensi dokumen dapat dilakukan dengan menyimpan dokumen pada penyimpanan lokal pada setiap *peers*.



Gambar 5.12: Ilustrasi Perbandingan Arsitektur *Peer-To-Peer* dan *Client-Server* Secara Berurutan

Drawback dari transmisi rendah pada aplikasi pengguna CRDT *client-server* ialah transmisi yang lebih tinggi dari sisi server. Gambar 5.12 menunjukkan bahwa banyaknya koneksi kanal data pada *peer-to-peer* tumbuh secara kuadratik. Total transmisi yang harus ditanggung oleh sebuah *peer* pada arsitektur ini secara matematis akan bertumbuh lebih cepat terhadap jumlah pengguna pada suatu kelompok jaringan. Selain itu, bila dilihat dari algoritma resolusinya, variasi OT memiliki muatan jaringan paling besar dibandingkan dengan variasi lain yang memengaruhi utilisasi CPU dan RAM-nya. *Resource* ini dipengaruhi oleh permintaan data masuk dan keluar yang cenderung lebih banyak. Dari aspek *lightweight*, arsitektur *client-server* pada OT tidak dipreferensi penggunaannya dan membutuhkan tinjauan kembali untuk dioptimisasi protokol pengiriman datanya. Selain itu, pemanfaatan variasi algoritma OT yang lebih kompleks dan optimal untuk dapat diteliti lebih lanjut untuk dapat menangani skenario perubahan intensif seperti skenario pertama.

BAB 6
PENUTUP

Bab ini memaparkan kesimpulan penelitian dan eksperimen yang telah dilakukan terhadap sistem yang telah dikembangkan. Bab ini juga memberikan rangkuman singkat tentang implementasi setiap variasi PeerToCP dan implikasi dari hasil evaluasi yang telah dianalisis. Selain itu, potensi pengembangan dan eksperimen lebih lanjut yang dapat diteliti di masa yang akan datang, disampaikan pula guna mengidentifikasi kelemahan dan memberikan kesempatan untuk meneliti topik atau sistem serupa yang lebih optimal.

6.1 Kesimpulan

Penelitian ini dibuat untuk mewujudkan aplikasi PeerToCP, yaitu sebuah editor kode kolaboratif yang menyediakan *shell* bersama dan bekerja dalam waktu nyata. Terdapat beberapa variasi arsitektur dan algoritma untuk menjaga konsistensi data dalam sistem terdistribusi yang digunakan dalam aplikasi ini, yaitu algoritma OT (*operational transformation*) dengan arsitektur *client-server*, struktur data CRDT (*conflict-free replicated data types*) dengan arsitektur *client-server*, dan juga CRDT yang digunakan bersamaan dengan arsitektur *peer-to-peer* berbasis WebRTC.

PeerToCP yang menggunakan *operational transformation* memanfaatkan sifat *transformation property* 1 untuk menjaga konsistensi setiap data pengguna pada suatu jaringan. Setiap perubahan akan direpresentasikan sebagai suatu operasi *update*. Klien dan server akan menyimpan rangkaian operasi ini. Setiap klien dapat melakukan sinkronisasi dokumen dengan klien lain pada jaringan dengan melakukan *pull update* terhadap server. Klien dapat menambahkan atau melakukan *push update* pada server dengan syarat versi terbaru dari server sudah dimiliki oleh klien saat ini. Bila hal tersebut belum dipenuhi, maka klien akan mendapatkan operasi-operasi terbaru pada server, dan menerapkan algoritma *operational transformation* pada semua operasi lokal yang belum diterima oleh server. *Shell* bersama yang disediakan pada aplikasi ini menggunakan konsep yang serupa dengan editor kodenya, namun *operational transformation*-nya dianggap memiliki kompleksitas waktu konstan karena setiap operasinya merupakan operasi *append* pada sebuah *array*.

Selain algoritma OT, terdapat pula variasi CRDT yang pada dasarnya merupakan pengembangan lanjutan dari algoritma OT dengan struktur data tambahan. Dalam penelitian ini, PeerToCP menggunakan *library* Yjs yang mengimplementasikan CRDT YATA. Setiap karakter pada struktur data ini dianggap memiliki *id* yang berbeda. Kemudian YATA menggunakan pendekatan dengan membuat sebuah struktur barisan linear dalam menyimpan karakternya. Posisi sebuah karakter akan direpresentasikan menjadi sebuah hubungan mendahului dan mengikuti karakter lain. Struktur data ini kemudian dapat dihubungkan dengan sebuah *provider* jaringan *client-server* yang menggunakan *library* YWebSocket berbasis WebSocket. Selain itu, terdapat pula variasi CRDT dengan arsitektur jaringan *peer-to-peer* yang menggunakan *library* YWebRTC berbasis WebRTC, serta memanfaatkan WebSocket untuk berkomunikasi dengan *signalling server*-nya. Kedua provider jaringan ini memberikan abstraksi bagi Yjs untuk dapat berkomunikasi dengan pengguna lain dalam jaringan dan melakukan proses sinkronisasi dokumen.

Pada aplikasi PeerToCP, setiap variasi yang dikembangkan telah dibuktikan kebenarannya secara teoritis oleh pengembang kodenya dan secara empiris melalui empat skenario yang dilakukan pada penelitian ini. Variasi utamanya dengan struktur data CRDT *peer-to-peer* menghasilkan performa latensi yang paling baik, serta kebutuhan *resource* yang lebih rendah pada aplikasi penggunanya. Variasi *operational transformation* dari aplikasi mengalami permasalahan transmisi jaringan dengan *bandwidth* terlalu besar karena protokol dan konkurensi *update* yang kurang optimal. Hal ini terjadi saat setiap klien pada suatu kelompok jaringan terus-menerus melakukan *push update* secara bersamaan. Server cenderung tidak dapat menerima operasi *update* pada suatu klien dengan operasi-operasi lokalnya yang sudah tertumpuk. Hal ini dikarenakan server mengutamakan penambahan versi yang dilakukan oleh klien lain dengan data *push update* yang lebih kecil.

Setiap variasi memiliki kelebihan dan kekurangan masing-masing pula baik dari aspek *lightweight* pada server maupun pada aplikasi pengguna. Berdasarkan skenario-skenario pengujian pada eksperimen ini, penggunaan untuk skala $n \leq 8$ dan setiap pengguna berada pada jarak dekat, PeerToCP dengan CRDT *peer-to-peer* berbasis WebRTC merupakan pilihan optimal. Alasan lainnya ialah karena muatan pada servernya jauh lebih rendah bila dibandingkan dengan arsitektur *client-server*. Pada versi *peer-to-peer* setiap komputasi akan dilakukan oleh pengguna masing-masing, sehingga *signalling server* pada arsitektur ini hanya digunakan untuk menangani pengguna yang terhubung dan terputus pada suatu

jaringan.

Pada kasus penggunaan pada kelompok jaringan yang lebih besar, variasi CRDT *client-server* dapat dipertimbangkan karena pertumbuhan transmisi data terhadap banyaknya klien yang lebih rendah. Selain itu, variasi ini juga dapat dipertimbangkan saat beberapa pengguna dalam jaringan kesulitan melanjutkan koneksi WebRTC karena jaraknya berjauhan atau adanya mekanisme jaringan yang menggagalkan koneksi untuk terhubung ke *peer* lain secara langsung.

6.2 Saran

Berdasarkan hasil penelitian ini, terdapat potensi pengembangan sistem lanjutan untuk membuat sebuah jaringan adaptif tergantung dari keadaannya. Potensi ini dapat menjadi salah satu solusi dalam mengoptimisasi layanan yang lebih *reliable* atau dapat diandalkan bagi semua penggunanya. Dari sisi algoritma dalam memastikan kesamaan replika data pada sebuah jaringan, variasi *operational transformation* atau CRDT lain yang lebih optimal dapat diteliti lebih lanjut untuk menggantikan yang ada pada variasi PeerToCP saat ini. Variasi ini diharapkan dapat mengoptimalkan latensi dan menurunkan penggunaan sumber daya yang dibutuhkan oleh sistem aplikasi. Salah satu metode yang berpotensi untuk mengoptimisasi variasi CRDT lebih lanjut ialah memodifikasi struktur data CRDT *map* atau *dictionary* yang digunakan untuk menyimpan replika *shell*. Struktur data CRDT ini seharusnya dapat memanfaatkan sifat bahwa data hanya akan dimasukkan ke ujung *array* saja tanpa ada proses penghapusan atau pengubahan pada indeks lain.

Selain algoritma, aspek jaringan dan teknologi web juga memiliki kesempatan pengembangan yang perlu mendapat perhatian. Beberapa teknologi baru seperti HTTP/3 yang merupakan versi terbaru dari HTTP pada saat penelitian ini dilakukan, menyediakan mekanisme WebTransport yang dapat diteliti lebih lanjut untuk menggantikan protokol WebSocket. WebTransport ditujukan untuk menyediakan antarmuka pemrograman yang lebih baik dan memiliki semua fitur yang dimiliki oleh WebSocket dengan latensi yang lebih rendah. Eksperimen lebih lanjut untuk menguji skalabilitas jumlah pengguna yang lebih besar dapat dilakukan dengan mengubah sistem pengujian tanpa perlu mengakses antarmuka secara *end-to-end* . Selain itu, pengembangan *frontend* dan aspek HCI (*Human-Computer Interaction*) juga dapat ditingkatkan dan diteliti dalam aplikasi PeerToCP.

Aspek performa dan pengalaman pengguna lebih lanjut dapat diperluas ke aplikasi

web yang dapat diakses tanpa perlu menggunakan aplikasi desktop seperti Electron, namun dengan *drawback* tidak dapat menjadi *host* untuk menyediakan *shell* yang dapat digunakan bersama oleh setiap pengguna dalam jaringan. Beberapa teknologi serta bahasa pemrograman lain yang memiliki performa lebih baik dan penggunaan *resource* lebih ringan dibandingkan Node.js dan Chromium pada Electron juga dapat diteliti untuk menggantikan *framework* aplikasi saat ini. Tauri dan Qt menjadi salah satu teknologi alternatif yang menjadi pertimbangan penulis dalam mengembangkan aplikasi PeerToCP.

DAFTAR REFERENSI

Adeyeye, M., Makitla, I., & Fogwill, T. (2013). Determining the signalling overhead of two common webrtc methods: Json via xmlhttprequest and sip over websocket. In *2013 africon* (pp. 1–5).

Aho, A., Sethi, R., & Ullman, J. (1985). Compilers: Principles, techniques, and tools.

Alvestrand, H. T. (2021a, January). *Transports for WebRTC* (No. 8835). RFC 8835. RFC Editor. Diakses dari https://www.rfc-editor.org/info/rfc8835 doi: 10.17487/RFC8835

Alvestrand, H. T. (2021b, January). *WebRTC MediaStream Identification in the Session Description Protocol* (No. 8830). RFC 8830. RFC Editor. Diakses dari https://www.rfc-editor.org/info/rfc8830 doi: 10.17487/RFC8830

Arefin, S. S., Azad, I., & Kabir, H. (2013). Modified sack-tcp and some application level techniques to support real-time application. *International Journal of Electrical and Computer Engineering (IJECE)*, *6*, 105–114.

Attiya, H., Burckhardt, S., Gotsman, A., Morrison, A., Yang, H., & Zawirski, M. (2016). Specification and complexity of collaborative text editing. In *Proceedings of the 2016 acm symposium on principles of distributed computing* (pp. 259–268).

Belomestnykh, O. (2010). *A rebuilt, more real time Google documents.* Google Developers Blog. Diakses pada tanggal 15 September 2022 pukul 14.28, dari https://drive.googleblog.com/2010/04/a-rebuilt-more-real-time-google.html

Belshe, M., Peon, R., & Thomson, M. (2015). *Hypertext transfer protocol version 2 (http/2)* (Tech. Rep.).

Bishop, M. (2022, June). *HTTP/3* (No. 9114). RFC 9114. RFC Editor. Diakses dari https://www.rfc-editor.org/info/rfc9114 doi: 10.17487/RFC9114

Day-Richter, J. (2010). *What's different about the new google docs: Making collaboration fast.* Google Developers Blog. Diakses pada tanggal 15 September 2022 pukul 15.29, dari https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html

Dutton, S., et al. (2012). Getting started with webrtc. *HTML5 Rocks*, *23*.

Ellis, C. A., & Gibbs, S. J. (1989). Concurrency control in groupware systems. In

Proceedings of the 1989 ACM SIGMOD international conference on management of data - SIGMOD '89. ACM Press. Diakses dari https://doi.org/10.1145/67544.66963 doi: 10.1145/67544.66963

Fette, I., & Melnikov, A. (2011). *The websocket protocol* (Tech. Rep.).

Fielding, R., & Reschke, J. (2015). *Hypertext transfer protocol (http/1.1): Message syntax and routing, ietf rfc 7230.*

Fietze, M. (2017). *Http/2 streams: Is the future of websockets decided?* Faculty of Computer Science, TU Dresden. https://www. rn. inf. tu-dresden ....

Firefox. (2022). *Spidermonkey documentation.* Firefox. Diakses dari https://firefox-source-docs.mozilla.org/js/index.html

Frindell, A., Kinnear, E., & Vasiliev, V. (2022, July 6). *WebTransport over HTTP/3* (Internet-Draft No. draft-ietf-webtrans-http3-03). Internet Engineering Task Force. Diakses dari https://datatracker.ietf.org/doc/draft-ietf-webtrans-http3/03/ (Work in Progress)

Ganaputra, J., & Pardamean, B. (2015). Asynchronous publish/subscribe architecture over websocket for building real-time web applications. *Internetworking Indonesia*, *7*(2), 15–19.

Gentle, J. (2011). *ShareJS – Live concurrent editing in your app.* ShareJS. Diakses pada tanggal 16 September 2022 pukul 11.08, dari https://sharejs.org/

Google. (2022). *V8 documentation.* Google. Diakses dari https://v8.dev/docs

Harris, J. (2010). *What's different about the new Google Docs?* Google Developers Blog. Diakses pada tanggal 15 September 2022 pukul 15.21, dari https://drive.googleblog.com/2010/05/whats-different-about-new-google-docs.html

IntelliJ, I. (2011). the most intelligent java ide. *JetBrains [online].[cit. 2016-02-23]. Dostupné z: https://www. jetbrains. com/idea/# chooseYourEdition*.

Jennings, C., Hardie, T., & Westerlund, M. (2013). Real-time communications for the web. *IEEE Communications Magazine*, *51*(4), 20–26.

Jesup, R., Loreto, S., & Tüxen, M. (2021, January). *WebRTC Data Channels* (No. 8831). RFC 8831. RFC Editor. Diakses dari https://www.rfc-editor.org/info/rfc8831 doi: 10.17487/RFC8831

Kinder, K. (2013). Sublime text: one editor to rule them all? *Linux Journal*, *2013*(232), 2.

Kleppmann, M., Gomes, V. B., Mulligan, D. P., & Beresford, A. R. (2019). Interleaving

anomalies in collaborative text editors. In *Proceedings of the 6th workshop on principles and practice of consistency for distributed data* (pp. 1–7).

Kredpattanakul, K., & Limpiyakorn, Y. (2018). Transforming javascript-based web application to cross-platform desktop with electron. In *International conference on information science and applications* (pp. 571–579).

Krishnamurthy, B., Mogul, J. C., & Kristol, D. M. (1999). Key differences between http/1.0 and http/1.1. *Computer Networks*, *31*(11-16), 1737–1751.

Leibnitz, K., Hoßfeld, T., Wakamiya, N., & Murata, M. (2007). Peer-to-peer vs. client/server: Reliability and efficiency of a content distribution service. In *International teletraffic congress* (pp. 1161–1172).

Li, D., & Li, R. (2004). Preserving operation effects relation in group editors. In *Proceedings of the 2004 ACM conference on computer supported cooperative work - CSCW '04*. ACM Press. Diakses dari https://doi.org/10.1145/1031607.1031683 doi: 10.1145/1031607.1031683

Lv, X., Cui, L., & Li, J. (2015). The research and design of real-time collaborative document management system. In *Proceedings of the 2015 3rd international conference on machinery, materials and information technology applications*. Atlantis Press. Diakses dari https://doi.org/10.2991/icmmita-15.2015.160 doi: 10.2991/icmmita-15.2015.160

Maly, R. J., Mischke, J., Kurtansky, P., & Stiller, B. (2003). Comparison of centralized (client-server) and decentralized (peer-to-peer) networking. *Semester thesis, ETH Zurich, Zurich, Switzerland*, 1–12.

Martin, J. L. (2020). *Conflict-free replicated data types (CRDT) for distributed JavaScript apps*. TL;DR. Diakses pada tanggal 18 September 2022 pukul 14.14, dari https://www.youtube.com/watch?v=M8-WFTjZoA0

Matthews, P., Rosenberg, J., & Mahy, R. (2010, April). *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)* (No. 5766). RFC 5766. RFC Editor. Diakses dari https://www.rfc-editor.org/info/rfc5766 doi: 10.17487/RFC5766

Miglani, S., & Shriram, S. (n.d.). Electron| build cross-platform desktop apps with javascript, html, and css.

Molli, P., Urso, P., Imine, A., et al. (2006). Tombstone transformation functions for ensuring consistency in collaborative editing systems. In *2006 international con-*

ference on collaborative computing: Networking, applications and worksharing (pp. 1–10).

Nédelec, B., Molli, P., Mostefaoui, A., & Desmontils, E. (2013). Lseq: an adaptive structure for sequences in distributed collaborative editing. In *Proceedings of the 2013 acm symposium on document engineering* (pp. 37–46).

Nicolaescu, P., Jahns, K., Derntl, M., & Klamma, R. (2016, November). Near real-time peer-to-peer shared editing on extensible data types. In *Proceedings of the 19th international conference on supporting group work.* ACM. Diakses dari https://doi.org/10.1145/2957276.2957310 doi: 10.1145/2957276.2957310

OpenJS Foundation. (2022). *What is electron?* OpenJS Foundation and Electron contributors. Diakses pada tanggal 19 November 2022 pukul 02.36, dari https://www.electronjs.org/docs/latest

Oster, G., Urso, P., Molli, P., & Imine, A. (2005). *Real time group editors without operational transformation* (Unpublished doctoral dissertation). INRIA.

Perkins, C., Westerlund, M., & Ott, J. (2021, January). *Media Transport and Use of RTP in WebRTC* (No. 8834). RFC 8834. RFC Editor. Diakses dari https://www.rfc-editor.org/info/rfc8834 doi: 10.17487/RFC8834

Petit-Huguenin, M., Nandakumar, S., Holmberg, C., Keränen, A., & Shpount, R. (2021, January). *Session Description Protocol (SDP) Offer/Answer Procedures for Interactive Connectivity Establishment (ICE)* (No. 8839). RFC 8839. RFC Editor. Diakses dari https://www.rfc-editor.org/info/rfc8839 doi: 10.17487/RFC8839

Pimentel, V., & Nickerson, B. G. (2012). Communicating and displaying real-time data with websocket. *IEEE Internet Computing*, *16*(4), 45–53.

Preguiça, N. (2018). *Conflict-free replicated data types: An overview.* arXiv. Diakses dari https://arxiv.org/abs/1806.10254 doi: 10.48550/ARXIV.1806.10254

Preguiça, N., Baquero, C., & Shapiro, M. (2018). *Conflict-free Replicated Data Types (CRDTs).* arXiv. Diakses dari https://arxiv.org/abs/1805.06358 doi: 10.48550/ARXIV.1805.06358

Ressel, M., Nitsche-Ruhland, D., & Gunzenhäuser, R. (1996). An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the 1996 ACM conference on computer supported cooperative work - CSCW '96.* ACM Press. Diakses dari https://doi.org/10.1145/240080.240305 doi: 10.1145/240080.240305

Reynolds, F. (2008). Web 2.0–in your hand. *IEEE Pervasive Computing*, *8*(1), 86–88.

Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011). Conflict-free replicated data types. In *Lecture notes in computer science* (pp. 386–400). Springer Berlin Heidelberg. Diakses dari https://doi.org/10.1007/978-3-642-24550-3_29 doi: 10 .1007/978-3-642-24550-3_29

Smith, Z. (2012). *Overview of operational transformation.* University of Minnesota. Diakses pada tanggal 18 September 2022 pukul 12.32, dari https://umm-csci.github .io/senior-seminar/seminars/spring2012/Smith.pdf

Sredojev, B., Samardzija, D., & Posarac, D. (2015). Webrtc technology overview and signaling solution design and implementation. In *2015 38th international convention on information and communication technology, electronics and microelectronics (mipro)* (pp. 1006–1009).

Srinivasan, R. (1995). *Rpc: Remote procedure call protocol specification version 2* (Tech. Rep.).

Stenberg, D. (2014). *Http2 explained* (Vol. 44) (No. 3). ACM New York, NY, USA.

Sun, C., & Ellis, C. (1998). Operational transformation in real-time group editors. In *Proceedings of the 1998 ACM conference on computer supported cooperative work - CSCW '98.* ACM Press. Diakses dari https://doi.org/10.1145/289444.289469 doi: 10.1145/289444.289469

Sun, C., Sun, D., Agustina, & Cai, W. (2019). *Real differences between OT and CRDT under a general transformation framework for consistency maintenance in co-editors.* arXiv. Diakses dari https://arxiv.org/abs/1905.01518 doi: 10.48550/ ARXIV.1905.01518

Sun, C., Xu, Y., & Ng, A. (2017, February). Exhaustive search and resolution of puzzles in OT systems supporting string-wise operations. In *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing.* ACM. Diakses dari https://doi.org/10.1145/2998181.2998252 doi: 10.1145/2998181.2998252

Sun, D., Sun, C., Ng, A., & Cai, W. (2019a). *Real differences between ot and crdt in correctness and complexity for consistency maintenance in co-editors.* arXiv. Diakses dari https://arxiv.org/abs/1905.01302 doi: 10.48550/ARXIV.1905.01302

Sun, D., Sun, C., Ng, A., & Cai, W. (2019b). *Real differences between OT and CRDT in building co-editing systems and real world applications.* arXiv. Diakses dari

https://arxiv.org/abs/1905.01517 doi: 10.48550/ARXIV.1905.01517

Sun, D., Xia, S., Sun, C., & Chen, D. (2004). Operational transformation for collaborative word processing. In *Proceedings of the 2004 ACM conference on computer supported cooperative work - CSCW '04*. ACM Press. Diakses dari https://doi.org/10.1145/1031607.1031681 doi: 10.1145/1031607.1031681

Thompson, S. (2015). *Smarter garbage collection for smoother browsing and less memory usage.* Chromium Developers Blog. Diakses pada tanggal 12 November 2022 pukul 16.24, dari https://blog.chromium.org/2015/12/smarter-garbage-collection-for-smoother.html

Tilkov, S., & Vinoski, S. (2010). Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, *14*(6), 80–83.

Vidot, N., Cart, M., Ferrié, J., & Suleiman, M. (2000). Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the 2000 ACM conference on computer supported cooperative work - CSCW '00*. ACM Press. Diakses dari https://doi.org/10.1145/358916.358988 doi: 10.1145/358916.358988

Weiss, S., Urso, P., & Molli, P. (2009). Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks. In *2009 29th ieee international conference on distributed computing systems* (pp. 404–412).

Xu, Y., & Sun, C. (2016, March). Conditions and patterns for achieving convergence in OT-based co-editors. *IEEE Transactions on Parallel and Distributed Systems*, *27*(3), 695–709. Diakses dari https://doi.org/10.1109/tpds.2015.2412938 doi: 10.1109/tpds.2015.2412938

LAMPIRAN

LAMPIRAN 1: KODE DAN IMPLEMENTASI APLIKASI

Setiap kode, implementasi aplikasi pengguna, serta eksperimen pengujian pada penelitian ini dapat diakses pada tautan repositori Github sebagai berikut `https://github.com/hockyy/peertocp`. Setiap variasi dari PeerToCP dipisahkan berdasarkan *branch*: `crdt-cs` yang merupakan variasi CRDT dengan arsitektur *client-server*, `ot-cs` yang merupakan variasi *operational transformation* dengan arsitektur *client-server*, serta `crdt-p2p` yang merupakan variasi CRDT dengan arsitektur *peer-to-peer*. Panduan untuk menjalankan kembali pengujian dan skenarionya terdapat pada bagian README.md dari branch `crdt-p2p` yang ditampilkan sebagai *branch* utama dari *repository*. Implementasi dari server dan modifikasi *provider* koneksi yang digunakan pada branch masing-masing dapat diakses pada tautan repositori Github:

- `crdt-cs`, dapat diakses pada `https://github.com/hockyy/y-websocket`;
- `crdt-p2p`, dapat diakses pada `https://github.com/hockyy/y-webrtc`;
- `ot-cs`, dapat diakses pada `https://github.com/hockyy/peertocp-ot-server`.

Dalam melakukan eksperimen, setiap *instance* diinisialisasi dengan beberapa aplikasi dan pengaturan melalui perintah-perintah sebagai berikut.

```
sudo apt install -y git wget screen nginx python-is-python3 g++ make
sudo apt install -y build-essential clang libdbus-1-dev libgtk2.0-dev \
                    libnotify-dev libgnome-keyring-dev libgconf2-dev \
                      libasound2-dev libcap-dev libcups2-dev
 ↪  libxtst-dev \
                      libxss1 libnss3-dev gcc-multilib g++-multilib
 ↪  libasound2 xvfb \
export DISPLAY=192.168.0.5:0.0
curl https://my-netdata.io/kickstart.sh > /tmp/netdata-kickstart.sh &&
 ↪  sh /tmp/netdata-kickstart.sh
curl -o-
 ↪  https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.2/install.sh |
 ↪  bash
source ~/.bashrc
```

```
nvm install 16
nvm use 16
git clone [URL]

# Menggunakan xvfb karena debian tidak ada desktop
xvfb-run npm start
```

Data diambil dan diproses pada perangkat lokal dengan *script* pengunduhan log hasil evaluasi sebagai berikut.

```
netd () {
    curl "http://$1:19999/api/v1/data?chart=apps.mem&dimension=node \
          &after=$2&points=0&group=average&gtime=0 \
          &timeout=0&format=csv&options=seconds" \
          > mem-$3.csv
    curl "http://$1:19999/api/v1/data?chart=system.ip \
          &after=$2&points=0&group=average&gtime=0 \
          &timeout=0&format=csv&options=seconds" \
          > network-$3.csv
    curl "http://$1:19999/api/v1/data?chart=apps.cpu&dimension=node \
          &after=$2&points=0&group=average&gtime=0 \
          &timeout=0&format=csv&options=seconds" \
          > cpu-$3.csv
}

scpd () {
    scp -r hocky@$1:~/peertocpnext/out/ ./$2
}
```

Setiap server menggunakan NGINX dengan konfigurasi sebagai berikut.

```
location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    # try_files $uri $uri/ =404;
```

```
    proxy_pass http://127.0.0.1:3000;

    proxy_http_version  1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

Hasil visualisasi dan interpretasi dari eksperimen yang disampaikan pada penelitian ini dapat diakses mellaui tautan repositori Github sebagai berikut `https://github.com/hockyy/peertocp-benchmark`.