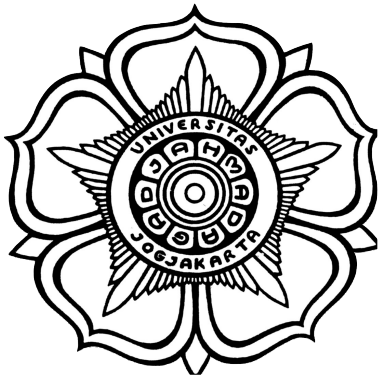
LAPORAN PROYEK AKHIR

PENERAPAN DAN ANALISIS HTTP/2 *SERVER-SENT EVENTS* DAN WEBSOCKET UNTUK *WEB APPLICATION* PADA SISTEM RUMAH PINTAR

ANALYSIS AND IMPLEMENTATION HTTP/2 SERVER-SENT EVENTS AND WEBSOCKET FOR WEB APPLICATION ON SMART HOME SYSTEM



Diajukan oleh :

MUHAMMAD RUSMINTO HADIYONO

15/386767/SV/10153

PROGRAM SARJANA TERAPAN TEKNOLOGI REKAYASA INTERNET

DEPARTEMEN TEKNIK ELEKTRO DAN INFORMATIKA

SEKOLAH VOKASI

UNIVERSITAS GADJAH MADA

YOGYAKARTA

2019

USULAN PROYEK AKHIR YANG DIAJUKAN KEPADA
PROGRAM SARJANA TERAPAN TEKNOLOGI REKAYASA INTERNET
SEKOLAH VOKASI UNIVERSITAS GADJAH MADA

1. JUDUL PROYEK AKHIR	:	Penerapan dan Analisis HTTP/2 *Server-Sent Events* dan WebSocket untuk W*eb Application* pada Sistem Rumah Pintar *Analysis and Implementation HTTP/2 Server-Sent Events and WebSocket for Web Application on Smart Home System*
2. PENYUSUN	:	Muhammad Rusminto Hadiyono
3. DOSEN PEMBIMBING I		
a. Nama Lengkap	:	Muhammad Arrofiq, S.T., M.T., Ph.D.
b. NIP	:	197311271999031001
4. TEMPAT PENELITIAN	:	Ruang Layanan Internet Teknologi Rekayasa Internet UGM.

Yogyakarta, 5 April 2019

Disetujui Oleh :

Dosen Pembimbing	Penyusun
Muhammad Arrofiq, S.T., M.T., Ph.D.	Muhammad Rusminto Hadiyono
NIP. 197311271999031001	NIM. 15/386767/SV/10153

Mengetahui,
Ketua Program Studi Teknologi Jaringan

Muhammad Arrofiq, S.T., M.T., Ph.D.
NIP. 197311271999031001

INTISARI

PENERAPAN DAN ANALISIS HTTP/2 *SERVER-SENT EVENTS* DAN WEBSOCKET UNTUK *WEB APPLICATION* PADA SISTEM RUMAH PINTAR

Akses internet yang cepat dan mudah didapatkan dapat mempermudah masyarakat untuk menerapkan teknologi *Internet of Things.* Bentuk penerapan yang sederhana dari *Internet of Things* yang bisa dimanfaatkan masyarakat adalah rumah pintar. Salah satu hal yang sering menjadi kendala dalam penerapan rumah pintar adalah cara mereka agar bisa terhubung dengan perangkat apapun di rumahnya melalui internet. Untuk terhubung dengan internet, setidaknya pengguna harus memiliki atau menyewa sebuah *web server* terlebih dahulu. Selain itu, pengguna harus tahu metode pengiriman apa yang diperlukan untuk menerima atau mengirimkan data dari *web browser* menuju mikrokontroler secara *real-time*. Terdapat berbagai metode pengiriman yang dapat diterapkan untuk kasus ini. WebSocket dan SSE adalah contoh beberapa protokol atau teknologi yang dapat dmanfaatkan untuk kasus *real-time,* namun yang manakah dari keduanya yang sesuai untuk digunakan di rumah pintar masih memerlukan pembahasan lebih lanjut. Maka dari itu pada proyek akhir ini dilakukan perbandingan teknologi SSE dengan Webscoket pada sistem pengendali rumah pintar berbasis *web*. Parameter yang diujikan adalah *response time* dan presentase penggunaan CPU.

Kata Kunci : *Internet of Things*, WebSocket, *Server-Sent Events*, MQTT, *smarthome.*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam menghadapi era Industri 4, Indonesia sudah memiliki banyak perkembangan di bidang teknologi dibandingkan era sebelumnya terutama di dalam pemanfaatan internet. Hal ini terlihat dari data statistik pengguna internet yang diterbitkan oleh Asosiasi Penyelenggara Jasa Internet Indonesia, pada tahun 2016 telah terdapat 132,7 juta pengguna sedangkan pada tahun 2017 sudah terdapat 143,26 juta pengguna dari total populasi penduduk Indonesia sebanyak 262 juta orang (APJII, 2017). Pesatnya pertumbuhan pengguna internet disebabkan oleh semakin cepatnya proses pengiriman data melalui internet serta semakin mudahnya cara untuk mendapatkan akses internet. Hal ini pula yang mendorong semakin beragamnya perangkat yang mampu terhubung dan saling terintegrasi atau lebih dikenal dengan istilah *Internet of Things*.

Konsep yang paling penting dari *Internet of Things* adalah mengintegrasikan semua hal yang ada di dunia nyata ke dalam dunia *digital* (Kwan et al. 2016). Salah satu penerapan dari *Internet of Things* adalah pengendalian rumah pintar melalui *web browser*. Hal yang perlu diperhatikan dalam pembuatan rumah pintar adalah kecepatan transaksi informasi, yang mana sebaiknnya memiliki nilai *response time* serendah mungkin (Saito and Menga 2015). Untuk mendapatkan nilai *response time* yang rendah, dibutuhkanlah infrastruktur jaringan yang bagus serta proses pengiriman data yang cepat dan tepat. Walaupun demikian, penerapan rumah pintar seringkali menggunakan infrastruktur jaringan seadanya serta menggunakan dana seminimal mungkin.

Dalam proses pengiriman data menuju *web browser* dengan rentang waktu sependek mungkin, terdapat berbagai pilihan yang dapat diterapkan pada rumah pintar, seperti halnya *Polling*, *Long-Polling*, WebSocket dan *Server-Sent Events*. Dari beberapa pilihan tersebut, *Polling* dan *Long Polling* termasuk metode pengiriman data yang harus secara aktif meminta data ke *server*. Di dalam kasus *Polling*, *client* akan meminta data ke *server* secara terus-menerus dengan jeda pengiriman yang telah ditentukan. Berbeda dengan *Polling*, *client* dalam kasus *Long-Polling* harus menunggu tersedianya pesan yang akan dikirimkan dari *server* setelah permintaan data dilakukan. Teknologi *Polling* dan *Long-Polling* kurang cocok digunakan dalam rumah pintar karena keduanya membutuhkan *bandwith* dan *response time* yang besar, berbeda halnya dengan *WebSocket* ataupun *Server-Sent Events* (Souders 2009).

WebSocket serta *Server-Sent Events* memungkinkan *web browser* menerima data dari *server* tanpa perlu *request* data baru setiap ada data baru, sehingga data yang telah tersedia di *server* akan dapat langsung dikirimkan ke *web browser*. Dengan berkurangnya waktu yang dibutuhkan untuk mengirimkan data, keduanya mampu untuk lebih *real-time* dibandingkan ketika menggunakan metode *Polling* maupun *Long-Polling*.

Di sisi lain, proses pengiriman data pada mikrokontroler juga memiliki pengaruh pula terhadap nilai *response time* dari *web browser*. Maka dari itu, dibutuhkanlah metode pengiriman yang cepat dan tepat untuk *Machine-to-Machine*. Dari beberapa protokol *Machine-to-Machine*, protokol MQTT yang berarsitektur *publish-broker-subscribe* memiliki rata-rata *response time* paling rendah (Kayal and Perros 2017). Dengan menggabungkan arsitektur *publish-broker-subscribe* yang dimiliki oleh MQTT dan

kelebihan yang dimiliki *WebSocket* ataupun *Server-Sent Events*, proses pengiriman data dari mikrokontroler menuju *web browser* akan lebih cepat.

Selain dari kecepatan transaksi data, rumah pintar cenderung dibuat secara sederhana. Karena itu, *server* yang digunakan untuk rumah pintar seringkali memiliki spesifikasi CPU maupun memori yang lebih rendah. Salah satu faktor yang berpengaruh terhadap performa *server* adalah proses pengolahan data, termasuk metode pengiriman data.

Baik *WebSocket* maupun *Server-Sent Events* masih memerlukan penelitian lebih jauh untuk mencari teknologi mana yang memiliki nilai *response time* paling rendah serta menggunakan *resource* di *server* terendah pula ketika diterapkan di rumah pintar, dengan alasan itulah penelitian ini dilakukan.

1.2 Rumusan Masalah

Rumusan permasalahan pada penelitian ini adalah bagaimana cara menerapkan HTTP/2 *Server-Sent Events* ataupun WebSocket pada sistem rumah pintar berbasis website serta untuk mengetahui hasil perbandingan *response time* serta presentase penggunaan CPU pada HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE dan WebSocket.

1.3 Batasan Masalah

Beberapa batasan masalah yang akan dilakukan selama proses pelitian proyek akhir adalah sebagai berikut :

1. Piranti elektronik yang terpasang dengan mikrokontroler bukanlah piranti elektronik yang benar-benar digunakan pada kehidupan sehari-hari di dalam rumah.
2. Pengujian dilakukan dalam satu waktu pada hari yang sama menggunakan jaringan internet Indihome.

3. Dikarenakan kebutuhan pengiriman data dua arah dari *web browser* maupun dari *web API*, *Server-Sent Events* tidak diuji secara satu arah tetapi pengujian dilakukan secara dua arah dengan bantuan *request* POST sesuai dengan protokol yang digunakan.
4. Tidak membahas keamanan pada jaringan maupun data.

1.4 Tujuan Penelitian

Tujuan dari penelitian dalam proyek akhir ini adalah mengimplementasikan serta membandingkan metode pengiriman data melalui HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE serta WebSocket dari rumah pintar menuju *browser* pengguna beserta sebaliknya menggunakan parameter *response time* serta presentase penggunaan CPU.

1.5 Manfaat Penelitian

Manfaat yang dapat diambil dari penelitian dan pengerjaan proyek akhir ini adalah sebagai berikut :

1. Memberi informasi mengenai implementasi HTTP/2 *Server-Sent Events*, serta *WebSocket* dari *server API* menuju *web browser*.
2. Memberi informasi mengenai cara mudah mengawasi dan mengubah status perangkat dari jarak jauh.
3. Hasil perbandingan *response time* dari ketika *web browser* meminta sampai mendapatkan data dari *server API* dengan metode yang berbeda (HTTP/2 *Server-Sent Events* dan *WebSocket*).
4. Hasil perbandingan presentase penggunaan CPU server ketika diterapkan metode pengiriman yang berbeda (HTTP/2 *Server-Sent Events* dan *WebSocket*).

1.6 Sistematika Penulisan

Untuk menggambarkan secara menyeluruh mengenai masalah yang akan dibahas dalam laporan proyek akhir ini, maka dibuat sistematika penulisan yang terbagi dalam lima bab :

BAB I, PENDAHULUAN, memuat latar belakang masalah, perumusan masalah, batasan masalah, tujuan penulisan, kegunaan penulisan dan sistematika penulisan.

BAB II, TINJAUAN PUSTAKA, berupa uraian sistematis tentang informasi yang relevan dan mutakhir yang terkait dengan lingkup materi penelitian atau teknologi yang akan diterapkan. Uraian dalam tinjauan pustaka ini selanjutnya menjadi dasar teori yang digunakan oleh penulis dalam melaksanakan penelitian dan menyajikan argumentasi dalam pembahasan hasil penelitian.

BAB III, BAHAN DAN METODE PENELITIAN, memuat bahan, peralatan, tahapan penelitian, dan rancangan sistem serta analisis data yang ada pada penelitian ini.

BAB IV, ANALISIS HASIL DAN PEMBAHASAN, memuat semua temuan ilmiah yang diperoleh sebagai data hasil penelitian, atau hasil unjuk kerja prorotipe yang dibuat. Pada bagian ini peneliti menyusun secara sistematis disertai argumentasi yang rasional tentang hasil unjuk kerja yang diperoleh dari hasil penelitian.

BAB V, PENUTUP, memuat kesimpulan serta saran dari penelitian yang dilakukakan pada proyek akhir ini.

BAB II

TINJAUAN PUSTAKA

2.1 Mikrokontroler

Microprocessor telah banyak membantu pekerjaan manusia, baik dalam hal pengendalian suatu peralatan maupun pemantauan. *Microprocessor* dapat diibaratkan sebagai otak dari suatu komputer, yang berisi *Control Unit* (CU) dan *Aritmetic and Logic Unit* (ALU). Dalam penggunaannya, *microprocessor* yang dikenal sebagai *single-chip computer* memerlukan *chip* tambahan untuk dapat bekerja. Sebagai hasilnya, banyak *chip* yang perlu disatukan dan membutuhkan daya yang semakin besar dalam penggunaannya. Di samping itu dengan banyaknya *chip* yang digunakan, biaya yang diperlukan dalam pembuatan pun juga akan semakin mahal dan pemecahan masalah akan semakin sulit dikarenakan rumitnya sambungan antar *chip* (Ibrahim 2014).

Untuk menyelesaikan permasalahan tersebut dibuatlah mikrokontroler, yang dibuat dengan merangkai *microprocessor* bersama dengan *chip-chip* tertentu menjadi sebuah *chip* sehingga mempermudah dalam penggunaan maupun perawatannya. Sebuah mikrokontroler terdiri dari sebuah atau beberapa *microprocessor*, *memory*, ADC (*Analog-to-Digital Controller*), DAC (*Digital-to-Analog Controller*), *Parallel I/O interface*, *Serial I/O interface*, serta penghitung waktu. Dalam penerapannya, mikrokontroler telah dapat ditemukan pada berbagai peralatan elektronik yang telah digunakan sehari-hari seperti mesin cuci, *printer*, *keyboard* maupun beberapa komponen mesin mobil (Udayashankara and S Mallikarjunaswamy 2009).

Saat ini mikrokontroler telah mampu terhubung dengan jaringan internet. Dengan terhubungnya mikrokontroler dengan internet, suatu mikrokontroler mampu mengirim maupun menerima data melalui jaringan menuju sistem lain yang mampu mengelola data-data tersebut untuk ditampilkan ke berbagai antarmuka maupun disimpan. Konsep inilah yang dinamakan *Internet of Things.*

2.2 Konsep *Internet of Things*

Internet of Things merupakan penamaan atas suatu sistem yang menghubungkan suatu atau beberapa *smart object* dengan *smart object* lainnya ataupun dengan sistem informasi lainnya melewati jaringan IP (*Internet Protocol*) (Cirani et al. 2018). Setiap *smart object* terdiri dari *microprocessor*, perangkat komunikasi, sensor atau aktuator serta sumber energi listrik. *Microprocessor* berguna untuk memberikan kemampuan komputasi pada *smart object*. Perangkat komunikasi memungkinkan *smart object* untuk berkomunikasi dengan *smart object* lainnya ataupun sistem lainnya. Sensor atau aktuator menghubungkan *smart object* dengan lingkungannya, memungkinkan mereka untuk mengukur besaran fisis tertentu sampai halnya mengendalikan obyek tertentu. Sumber energi listrik dibutuhkan untuk menjalankan perangkat elektronik pada *smart object*, yang mana dapat berupa baterai ataupun dari sumber energi listrik lainnya (Vasseur et al. 2010). *Internet of Things* dapat diterapkan pada berbagai skenario seperti rumah pintar, *smart cities* serta pengolahan agrikultur (Cirani et al. 2018).

2.3 Rumah Pintar

Rumah pintar adalah istilah yang digunakan untuk sekumpulan perangkat yang digunakan dalam kehidupan sehari-hari yang saling terhubung dalam suatu jaringan. Perangkat yang terhubung bisa berupa lampu, kunci pintu, gerbang, pintu garasi, *DVD*

player, CCTV, sensor suhu, sensor asap sampai halnya *laptop* ataupun *server*. Dengan menerapkan rumah pintar, status atau informasi yang dimiliki dari setiap perangkat dapat diperoleh ataupun diubah melalui perangkat lainnya (Briere and Hurley 2011).

Dalam penerapannya, terdapat banyak pilihan protokol yang dapat digunakan untuk mengendalikan *smart object* yang berada di dalam rumah menuju *server*. Walaupun demikian, dalam penyusunan rumah pintar dibutuhkan protokol yang memiliki proses pengiriman yang cepat dan tepat. Protokol MQTT adalah salah satu protokol yang sering digunakan untuk keperluan pengiriman data *machine-to-machine*.

2.4 *Message Queuing Telemetry Transport*

Message Queuing Telemetry Transport (MQTT) adalah protokol yang ringan dan bekerja dengan sistem *publish-broker-subscribe*. Protokol MQTT bekerja di atas protokol *Transmission Control Protocol* / *Internet Protocol* (TCP/IP). MQTT sangat cocok digunakan untuk pengiriman data yang bersifat *real-time* (Hillar 2017). Selain itu, dalam penerapannya semakin singkat jeda pengiriman data melalui protokol MQTT semakin kecil nilai *delta time*-nya serta nilai integritas data yang dikirim dan diterima mencapai 100% (Rochman, Primananda, and Nurwasito 2017). Walaupun demikian, protokol MQTT tidak bisa digunakan secara langsung ke *web browser*, sehingga protokol MQTT perlu dibungkus dengan *WebSocket* atau dilewatkan ke *server* lain untuk mengubah protokol MQTT ke HTTP. Di dalam penerapannya, membungkus MQTT dengan *WebSocket* dapat diwujudkan dengan mudah. Namun dalam kasus-kasus tertentu, semisal diperlukannya *Application Programming Interface* (API) untuk proses penyimpanan, pengumpulan, serta pengolahan data maka pengubahan metode pengiriman dari MQTT menuju ke HTTP untuk ditampilkan ke *web browser* diperlukan. Selain HTTP masih banyak protokol yang dapat

digunakan untuk mengirimkan data dari *web API* menuju *web browser*. Protokol – protokol tersebut memiliki berbagai karakteristik yang berbeda, namun berdasarkan format data yang dikirimkan protokol-protokol tersebut dapat dikategorikan menjadi dua, yakni *Plain Text Protocol* dan *Binary Protocol.*

2.5 *Binary Protocol* dan *Plain Text Protocol*

Plain Text Protocol adalah protokol yang dapat dengan mudah dibaca oleh manusia, contohnya adalah HTTP/1.1 dan SMTP. Sedangkan *Binary Protocol* adalah protokol yang ditujukan untuk dibaca langsung oleh mesin dengan tujuan mempercepat proses penafsiran data. Contoh dari protokol yang termasuk kategori *Binary Protocol* adalah HTTP/2 serta *WebSocket* (Rhee and Hyun Yi 2015). Data yang dikirimkan melalui *Binary Protocol* cenderung lebih ringan daripada ketika dikirmkan melalui *Plain Text Protocol*. Hal ini dikarekanan data yang dikirimkan melalui *Binary Protocol* lebih berorientasi ke struktur data dibandingkan *Plain Text Protocol* yang lebih cenderung ke *Text String*. Dengan adanya susunan yang lebih mengutamakan struktur data, dapat memungkinkan pengiriman angka dalam bentuk *integer* bukan sebagai beberapa karakter dilakukan. Tidak hanya *integer* saja, tetapi hal ini juga berlaku untuk berbagai tipe data lainnya yang telah ditetapkan oleh suatu *Binary Protocol*. Dari kedua kategori tersebut, HTTP adalah protokol komunikasi yang termasuk ke dalam keduanya.

2.6 *Hypertext Transfer Protocol*

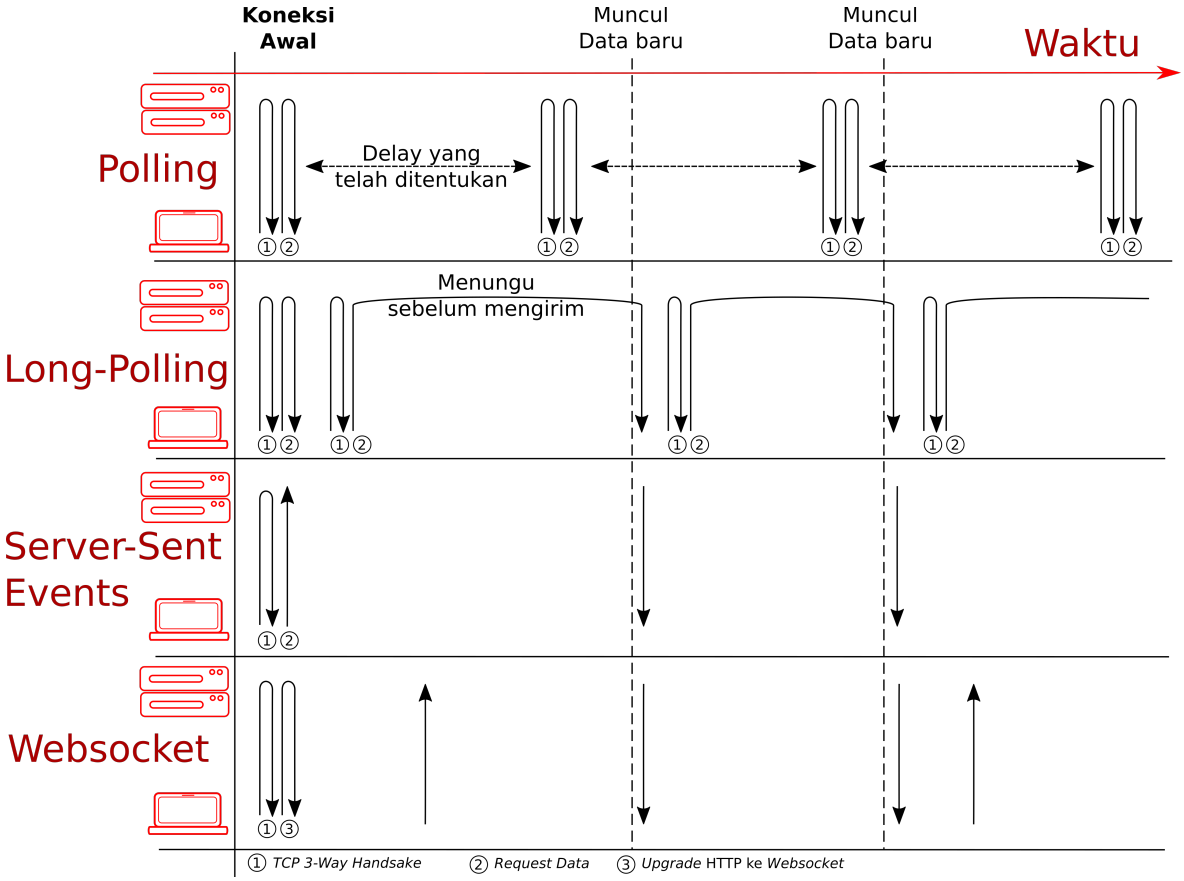
Hypertext Transfer Protocol (HTTP) adalah protokol komunikasi yang berguna untuk komunikasi antara *web browser* dengan *web server* yang telah digunakan oleh *World-Wide Web* sejak 1990. Protokol HTTP berada pada *application-layer* di dalam OSI *Layer.* Dalam perkembangannya, HTTP memiliki beberapa generasi. Diawali dengan HTTP/0.9 yang

berupa protokol sederhana untuk pengiriman *raw data* di *internet*. Setelah itu muncul generasi berikutnya yakni HTTP/1.0 yang memungkinkan pengiriman informasi dalam *format* seperti MIME. Sayangnya, HTTP/1.0 tidak sesuai dengan jaringan yang memakai *proxy*, *caching*, ataupun *virtual host* serta jaringan yang membutuhkan koneksi yang bertahan lama (Leach et al. 1999).

Kemudian munculah HTTP/1.1, HTTP/1.1 telah memperbaiki masalah yang ada pada HTTP/1.0. Dengan wajib menyertakan *Host header*, memungkinkan HTTP/1.1 untuk melakukan *virtual hosting* ataupun melayani beberapa pengguna yang berbeda pada sebuah alamat IP. Keunggulan dari HTTP/1.1 daripada HTTP/1.0 adalah munculnya OPTIONS *method*, *upgrade* pada *header*, pemampatan dengan *transfer-encoding* maupun *pipelining (Ludin and Garza 2017)*. Pada generasi ini pula, muncul WebSocket yang memanfaatkan kemampuan *upgrade* pada *header* HTTP/1.1.

Setelah HTTP/1.1 bertahan cukup lama, munculah HTTP/2 yang memungkinkan penggunaan jaringan yang lebih efisien dengan melakukan pemampatan *header* menggunakan HPACK . Selain itu, HTTP/2 mampu menangani *Head of Blocking* yakni kejadian ketika data yang diterima dari *server* harus mengantri untuk bisa dimuat pada halaman HTML. Hal ini mengakibatkan beberapa data yang dikirim maupun diterima dapat dilakukan dalam satu waktu tanpa saling menghalangi dengan kata lain HTTP/2 mampu melakukan *multiplexing*. HTTP/2 juga tidak membutuhkan koneksi tambahan untuk memungkinkan koneksi paralel dan hanya cukup menggunakan satu koneksi HTTP/2 (Peon and Ruellan 2015). Sebaga informasi tambahan, saat ini HTTP/2 masih harus menggunakan koneksi yang aman (TLS/SSL).

Dalam kasus pemantauan sensor maupun status aktuator terbaru dari *web browser*, dibutuhkan metode pengambilan data terbaru dari *server* secara terus-menerus. Pengambilan data terbaru secara terus-menerus dapat dilakukan menggunakan metode *Polling*, *Long-Polling*, *Server-Sent Events*, maupun WebSocket. Dari beberapa pilihan tersebut, perlu dipilih metode yang membutuhkan penggunaan memori dan CPU serendah mungkin dan juga tidak memakan banyak *bandwith*. Untuk cara kerja masing-masing metode dapat diamati pada Gambar 2.1.



Gambar 2.1 Berbagai metode pengiriman data ke *web browser*

Metode *Polling* atau yang lebih sering dikenal dengan 'AJAX' bekerja dengan cara *web browser* melakukan permintaan data dalam setiap kurun waktu yang telah ditentukan. Sehingga apabila tersedia data baru di sisi server, maka data tersebut tidak akan langsung

diterima oleh *web browser*. Dengan data yang tidak langsung diterima, dapat mengakibatkan kenaikan nilai *response time* yang dibutuhkan untuk setiap data baru yang diterima. Selain itu, dengan begitu banyaknya permintaan data untuk setiap data baru yang diterima, metode ini dapat memakan *bandwith* lebih banyak dibandingkan metode-metode lainnya.

Untuk mengatasi data yang tidak langsung diterima pada metode *Polling*, munculah metode *Long Polling*. Metode ini bekerja dengan melakukan permintaan data terlebih dahulu dan dilanjutkan dengan menunggu tersedianya data baru dari sisi *server*. Apabila data baru telah diterima, *web browser* akan melakukan permintaan data lagi sampai mendapatkan data terbaru dari *server*. Walaupun demikian, metode ini masih memakan banyak *bandwith*, apalagi dalam kondisi ketika *server* menyediakan data baru setiap beberapa *miliseconds.* Masalah lain yang timbul ketika menggunakan metode *Long-Polling* adalah ketika terjadi beberapa permintaan data dari satu *web browser* secara serentak, hal ini dapat menyebabkan data yang diterima saling tumpang tindih ataupun terjadinya duplikasi data secara berlebih. Selain itu, *Long-Polling* memperumit arsitektur server ketika digunakan pada beberapa server yang saling berbagi kondisi *session's client* (Abyl 2018). Metode lain yang dapat digunakan selain *Polling* dan *Long Polling* pada protokol HTTP adalah *Server-Sent Events.*

2.7 *Server-Sent Events*

Server-Sent Events memungkinkan *server* untuk mengirimkan data baru ke *web browser* setiap mucul data baru dari sisi *server* (*streaming*). Berbeda dengan *Long-Polling* maupun *Polling,* metode ini tidak perlu melakukan permintaan data untuk setiap data baru yang muncul pada *server* sehingga dapat mengurangi penggunaan *bandwith,* CPU serta

memori berlebih (Örnmyr and Appelqvist 2017). Setiap kali *web browser* mengirimkan permintaan data ke *server*, metode ini mampu membuka koneksi selama *server* tidak menghentikkannya. Selama itu pula, data baru yang tersedia di *server* dapat langsung dikirimkan ke *web browser* dalam bentuk potongan-potongan data (*chunk*).

Kemampuan lain yang dimiliki oleh *Server-Sent Events* adalah kemampuan untuk terhubung kembali apabila terjadi putus koneksi antara *web browser* dengan *server*. Namun karena itu pula, server tidak mampu mengetahui kapan web browser terputus tanpa mencoba mengirimkan data terlebih dahulu. Selain itu, *Server-Sent Events* pada *web browser* yang diterapkan menggunakan bahasa pemrograman Javascript (*EventSource object*) tidak mampu melakukan penambahan *field* pada *headers*, *field* yang digunakan akan selalu sama yakni hanya berisikan "Content-Type: text/event-stream". Tidak mampu mengubah *headers* berdampak pada ketidakmampuan *web browser* menggunakan *Authorization* yang berguna untuk memastikan keamanan pengirim data ataupun fungsi *field headers* lainnya. Selain itu, dalam penerapannya di HTTP/1.1, *Server-Sent Events* hanya terbatas memiliki enam koneksi yang terbuka dalam satu *web browser*. Namun, dengan dikeluarkannya HTTP/2 koneksi yang mampu terbuka di saat bersamaan semakin bertambah. Sebagai catatan tambahan, Internet Explorer tidak mendukung *Server-Sent Events* maupun *WebSocket* (Elman and Lavin 2014).

2.8 *WebSocket*

Protokol *WebSocket* memungkinkan komunikasi *full duplex* antara *web browser* dengan *server* melalui jaringan internet. Dalam penerapannya, protokol *WebSocket* perlu melakukan permintaan *upgrade* koneksi pada protokol HTTP/1.1. Hal ini dilakukan guna berganti jalur dari HTTP ataupun HTTPS menuju protokol *WebSocket*. Apabila *server*

memutuskan untuk *upgrade* koneksi, *server* akan mengirimkan pesan "101 Switching Protocols", namun jika *server* menolak maka permintaan akan diabaikan. Setelah protokol berpindah ke protokol *WebSocket*, *handsake* pembuka akan dilakukan. Pada *WebSocket*, *handshake* pembuka berupa membukanya suatu koneksi baru. Setelah terbukanya koneksi baru, *server* maupun *web browser* dapat saling bertukar pesan (MDN 2019).

Untuk menggunakan protokol *WebSocket* pada suatu aplikasi, dibutuhkanlah *WebSocket API*. *WebSocket API* dikembangkan oleh *World Wide Web Consortium* (W3C) sedangkan untuk protokol *WebSocket* dikembangkan oleh *Internet Engineering Task Force* (IETF). Dengan menggunakan *WebSocket API*, tindakan untuk membuka dan menutup koneksi, mengirim pesan, maupun menangkap pesan dari *server* melalui protokol *WebSocket* dapat dilakukan (Wang, Salim, and Moskovits 2013). Salah satu kelebihan protokol *WebSocket* adalah memiliki rata-rata *response time* lebih rendah dalam kondisi pengguna *resource* CPU yang terus naik (Kayal and Perros 2017). Walaupun demikian penggunaan protokol *WebSocket* secara langsung tanpa menggunakan (TLS/SSL) rentan terhalangi oleh *proxy server*. Namun hal ini tidak berlaku pada *Secure WebSocket* maupun HTTP/2 dikarenakan data telah terenkripsi dengan TLS/SSL (Ramli, Jarin, and Suryadi 2018).

Dalam penerapannya baik WebSocket maupun *Server-Sent Events*, metode yang digunakan untuk pengiriman data dapat mempengaruhi nilai *response time*. Penelitian untuk menguji *delay* (*one-way trip*) antara *WebSocket* dengan *Server-Sent Events* juga pernah dilakukan dan diperoleh hasil bahwa adalah rata-rata *delay* dari penggunaan *Server-Sent Events* lebih kecil dibandingkan ketika menggunakan *WebSocket* (Muhammad, Yahya, and Basuki 2018).

2.9 *Response Time*

Response Time merupakan parameter yang digunakan untuk mengukur waktu yang dibutuhkan oleh *web browser* untuk mendapatkan balasan dari *server*. Nilai *response time* dapat dipengaruhi oleh banyak hal. Beberapa faktor yang mempengaruhi nilai *response time* adalah infrastruktur jaringan, lama pengolahan data maupun *web browser* yang digunakan (Estep 2013). Dalam penelitian ini, menghitung *response time* berarti menghitung lamanya waktu yang dibutuhkan dari saat pengguna mengubah status perangkat dari *web browser* sampai mendapatkan status perangkat terbaru dari mikrokontroler.

Tabel 2.1 Ringkasan tinjauan pustaka

No	Judul Penelitian	Penulis & Tahun	Metode	Tipe Penelitian	Pengiriman Data	Kesimpulan
1	*A Comparison of IoT Application Layer Protocols Through a Smart Parking Implementation*	(Paridhika Khayal, dkk. 2017)	Membandingkan *response time* untuk protokol MQTT, CoAP, XMPP dan MQTT melalui WebSocket	Simulasi	Ubuntu 14.04 (MQTT, CoAP, XMPP dan MQTT-WS)	Ketika penggunaan *resource server* dinaikkan, *response time* WebSocket relatif tetap
2	*A Real-time Application Framework for Speech Recognition Using HTTP/2 and SSE*	(Kalamullah Ramli, dkk. 2018)	Membandingkan HTTP/2 SSE dan WebSocket dalam penerapan *Speech Recognition* pada ns-3	Simulasi	Ubuntu (HTTP/2 dan WebSocket)	Besar latensi aplikasi pada penggunaan HTTP/2 SSE serta WebSocket relatif sama serta WebSocket lebih rentan di-*block* oleh *proxy server* dibandingkan HTTP/2
3	Analisis Perbandingan Kinerja Protokol WebSocket dengan Protokol SSE pada Teknologi *Push Notification*	(Panser Brigade Muhammad, dkk. 2018)	Analisis Perbandingan Kinerja Protokol WebSocket dengan Protokol SSE pada Teknologi *Push Notification*	Purwarupa	Ubuntu – smartphone (WebSocket dan HTTP/1.1)	Rata-rata delay pada protokol SSE lebih kecil dibandingkan dengan WebSocket begitu juga dengan presentase penggunaan CPU

4	*Performance comparison of XHR polling, Long-Polling, Server-Sent Events and WebSockets*	(Oliver Örnmyr, dkk. 2017)	Membandingkan penggunaan memori dan CPU dari 100 perangkat virtual yang terhubung dengan server menggunakan *XHR Polling*, *Long-Polling* , *Server-Sent Events* dan WebSockets	Simulasi	Ubuntu (HTTP/1.1 dan Webscoket)	Penggunaan CPU, memori maupun bandwith pada SSE serta WebSocket lebih kecil dibandingkan dengan XHR *Polling* serta *Long-Polling*
5	*Mobile HTML5: Efficiency and Performance of WebSockets and Server-Sent Events*	(Elliot Estep, 2013)	Membandingkan performa browser ketika menggunakan WebSockets dan *Server-Sent Events* dalam berbagai jaringan *smartphone* (WiFi, 3G dan 4G)	Simulasi	Windows 7 (WebSocket dan HTTP/1.1)	Performa SSE maupun Websocket juga dipengaruhi oleh *web browser* serta konfigurasi jaringan yang digunakan

Penelitian yang dilakukan membandingkan *response time* serta presentase penggunaan CPU antara HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE dan WebSocket pada sistem kendali rumah pintar berbasis *web*. *Server* yang digunakan berupa Raspberry Pi 3 serta mampu diakses melalui internet.

3.2 Hipotesis

Berdasarkan kajian dari Tinjauan Pustaka, dapat dibuat hipotesis bahwa HTTP/2 *Server Sent Event* memiliki nilai *response time* terendah sedangkan HTTPS memiliki nilai *response time* tertinggi. Namun dari keempat metode yang digunakan, nilai penggunaan CPU keempatnya relatif sama.

BAB III

METODE PENELITIAN

Pada bab ini dijelaskan mengenai kebutuhan peralatan dan bahan serta perangkat lunak pendukung untuk melakukan pengembangan serta pengambilan data metode HTTP/1.1 *Server-Sent Events*, HTTP/2 *Server-Sent Events* dan *WebSocket* untuk *website* pada sistem rumah pintar. Selanjutnya dijelaskan juga mengenai metode penelitian dan skenario pengambilan data perbandingan.

3.1 Peralatan

Berikut merupakan daftar peralatan yang akan digunakan selama proses penelitian berlangsung :

1. Komputer sebagai *client* dengan spesifikasi:

   - CPU : Intel Celeron 1.50GHz x 4
   - Hardisk : 750 GB
   - RAM : 8 GB
   - OS : Linux Mint 18.2 Cinnamon 64-bit

2. Raspberry Pi 3 Model B sebagai *broker* sekaligus *web server* dan *server API* sebanyak 1 unit dengan spesifikasi:

   - CPU : 1.2 GHz quad-core ARM
   - *Memory* : 1 GB LPDDR2-900 SDRAM
   - USB Port : 4
   - Network : 10/100 Mbps Ethernet, 802.11 n Wireless LAN

3. NodeMCU 1 Unit dengan spesifikasi:

   - MCU : Xtensa Dual-Core 32-bit L106
   - Wifi : 802.11 b/g/n HT40
   - Typical Frequency : 160 MHz
   - Tipe ESP : ESP32

4. NodeMCU 1 Unit dengan spesifikasi:

   - MCU : Xtensa Single-Core 32-bit L106
   - Wifi : 802.11 b/g/n HT20
   - Typical Frequency : 80 MHz
   - Tipe ESP : ESP8266

5. *Access Point* sebanyak – 1 unit
6. Sensor DHT11 – 2 unit
7. LED Putih – 13 unit
8. Servo Motor SG90 – 2 unit
9. *Motor* DC 3 volt – 1 unit
10. *Cooling fan* DC 5 volt – 1 unit
11. Kabel Jumper

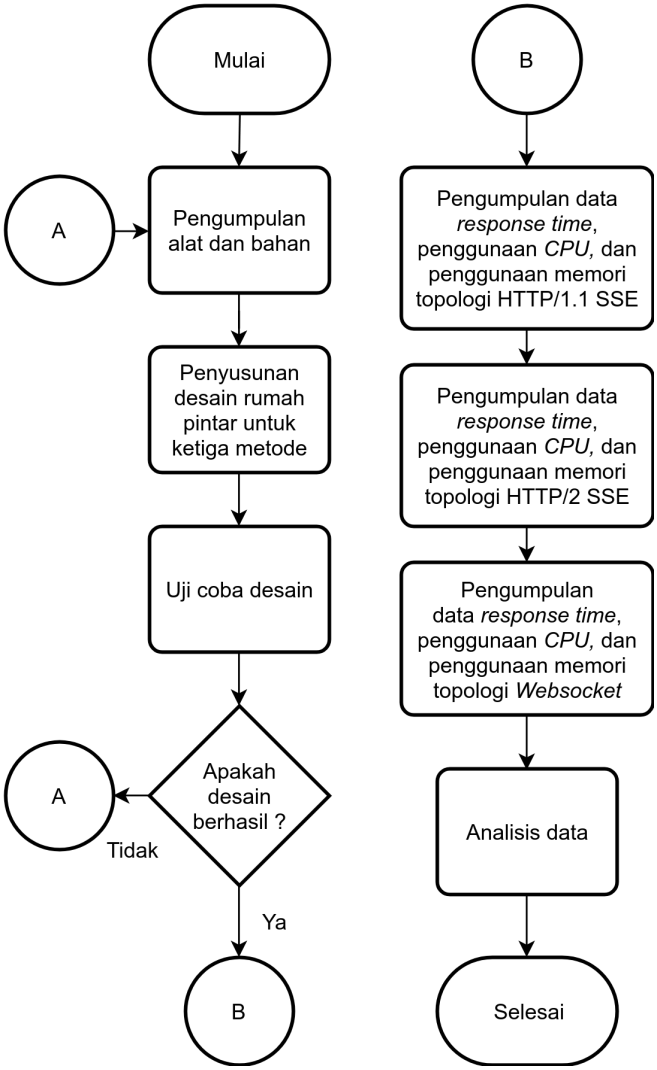
3.2 Bahan

Berikut merupakan daftar perangkat lunak yang akan digunakan selama penelitian ini berlangsung :

1. Mosquitto, sebagai broker untuk protokol MQTT
2. Node.js, sebagai *web server* yang khusus Javascript
3. Vue.js, sebagai javascript framework untuk pembuatan *web application*
4. OpenSSL, untuk pembuatan *public key* dan *private key* pada HTTPS serta HTTP/2
5. Serveo, untuk menjadikan *local server* mampu diakses secara publik
6. Google Chrome, untuk mengakses *website*

3.3 Tahapan Penelitian

Metode penelitian yang dilakukan penulis dalam penelitian ini dapat dilihat pada *flow chart* pada Gambar 3.1.



Gambar 3.1 Diagram alir penelitian

1. Tahap Instalasi.

   Tahap pertama yang dilakukan adalah menginstal semua perangkat lunak yang dibutuhkan. Adapun perangkat lunak yang di-*install* pada tahap ini adalah

Mosquitto, Node.js, Vue.js dan OpenSSL yang akan dipasang pada Raspberry Pi.

2. Tahap Pengembangan Sistem.

   Pada tahap pengembangan sistem, terdapat beberapa fase yang harus dilalui, yakni :

   a. Perancangan Sistem

      Fase perancangan aplikasi memiliki tujuan untuk memberikan gambaran umum mengenai bagaimana proses yang akan berjalan pada sistem yang akan diterapkan.

   b. Pembuatan Sistem

      Fase pembuatan sistem bertujuan untuk mengimplementasikan *WebSocket API* serta *Server-Sent Events API* baik pada sisi *client* maupun *server*. Selain itu, pada fase ini akan diterapkan pula penerapan MQTT di sisi mikrokontroler.

3. Tahap Pengambilan Data

   Di dalam tahap ini akan dilakukan proses pengambilan data untuk metode HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE serta Werbsocket yang telah diterapkan di dalam tahap pengembangan sistem. Parameter yang akan digunakan untuk pengambilan data adalah presentase penggunaan CPU serta *response time* untuk setiap metode yang digunakan. Pengambilan data *response time* dilakukan dengan menghitung waktu antara pengubahan status perangkat yang dilakukan dari *web browser* sampai dengan mendapatkan balasan status perangkat terbaru dari mikrokontroler.

4. Tahap Analisis.

   Analisis merupakan tahapan paling akhir dalam penelitian ini. Pada tahap ini hasil pengambian data yang telah dilakukan pada tahap sebelumnya akan dikumpulkan, diurai, dibedakan dan dipilah untuk selanjutnya digolongkan dan dikelompokkan berdasarkan kriteria tertentu kemudian dibuat suatu kesimpulan dari hasil yang didapat.

3.4 Perancangan Topologi dan Model

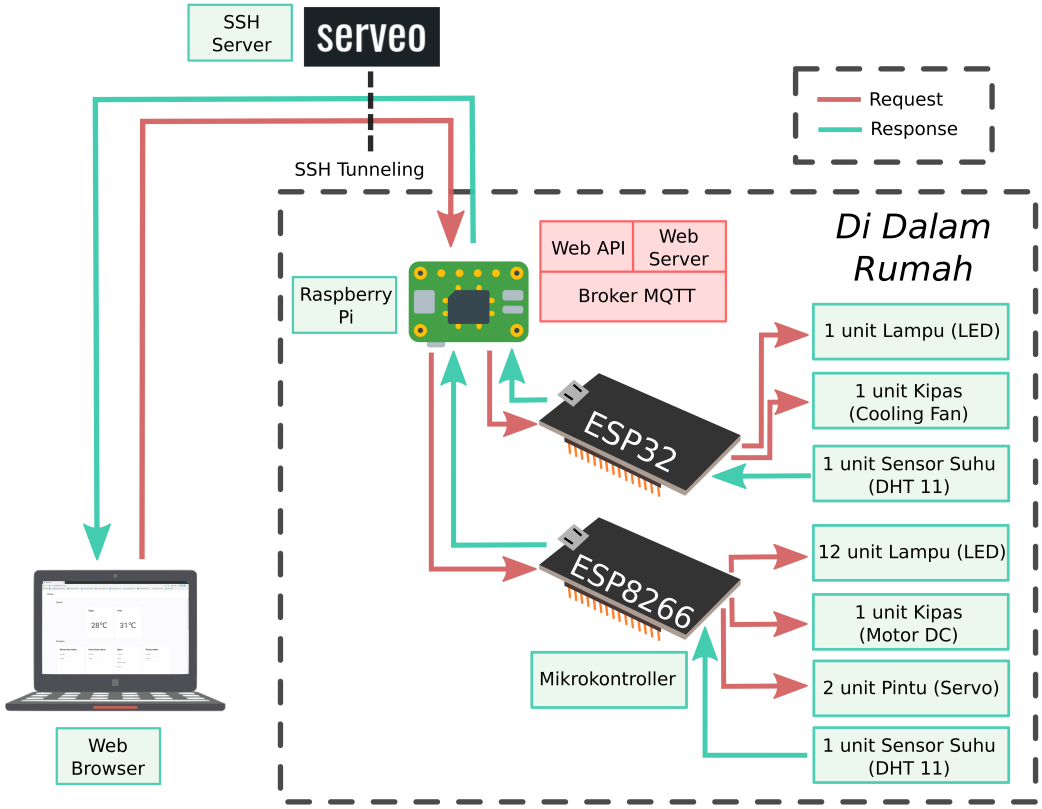
Dalam tahap pengembangan aplikasi, dilakukan dua tahap perancangan, yakni perancangan topologi dan perancangan model.

3.4.1. Perancangan Topologi

Perancangan topologi dibuat dengan memperhatikan gambaran umum proses yang terjadi pada sistem beserta hubungan antar komponen di dalamnya. Perancangan topologi dapat dibagi menjadi perancangan topologi perangkat serta perancangan topologi data. Perancangan topologi perangkat berguna untuk mendeskripsikan alur kerja sistem berdasarkan perangkat yang digunakan maupun dilalui sedangkan perancangan topologi data untuk menggambarkan lalu lintas data yang terjadi pada sistem.

1. Perancangan Topologi Perangkat

Berdasarkan perangkat yang digunakan, terdapat tiga komponen utama yang dibutuhkan supaya sistem mampu berjalan yakni *web browser*, *server* serta mikrokontroler. Untuk lebih lengkapnya dapat diamati pada Gambar 3.2.



Gambar 3.2 Bagan Topologi Perangkat

Web browser merupakan komponen yang berperan sebagai antarmuka pada sistem yang akan dibuat. Data berupa HTML, CSS maupun Javascript yang akan ditampilkan pada *web browser* berasal dari *web server.* Untuk data yang berupa JSON maupun teks seperti halnya suhu berasal dari *web API*.

Komponen *server* dapat dibagi menjadi tiga bagian, yakni *web server, web API*, serta MQTT *Broker*. *Web server* berperan untuk mengirimkan data *web application* yang berupa HTML, CSS maupun javascript. *Web application* selanjutnya akan ditampilkan di *web browser* pengguna. Untuk pembuatan *web application* dapat dilakukan dengan berbagai metode. Dalam penelitian ini, *web application* dibuat menggunakan *javascript framework* yakni "Vue.js".

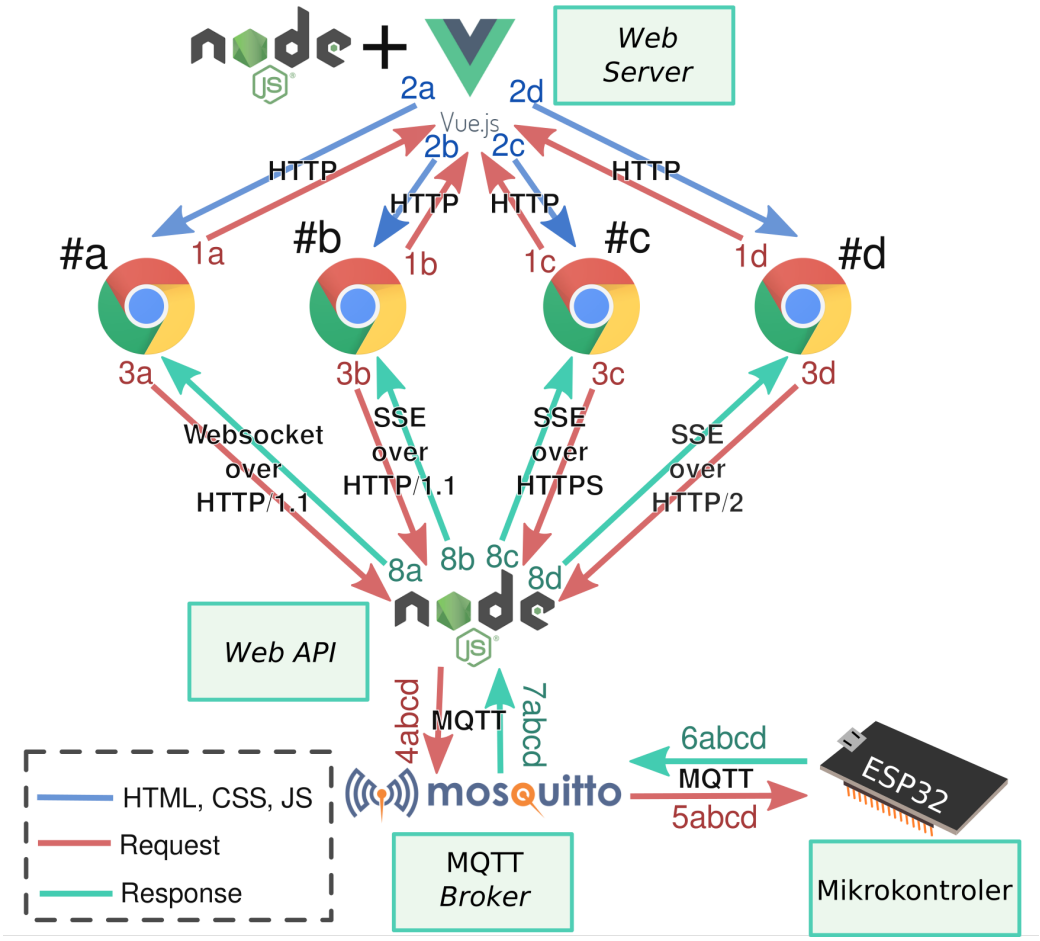
Selanjutnya, terdapat bagian *web API* yang bertugas untuk mengolah data yang berasal maupun menuju MQTT *Broker*. Baik web API maupun web server, keduanya bekerja dengan menggunakan perangkat lunak Node.js. Kemudian data yang telah masuk ke MQTT *Broker* akan diteruskan menuju mikrokontroler .

Dalam penerapannya, *server* tanpa menggunakan Serveo hanya dapat diakses pada jaringan lokal. Serveo adalah *server* SSH yang digunakan untuk *tunneling* dari suatu komputer sehingga mampu untuk diakses dari luar jaringan. Terdapat pula pihak ketiga yang serupa dengan Serveo, yakni Ngrok. Pemilihan Serveo daripada Ngrok didasarkan pada biaya yang dikeluarkan. Ngrok membutuhkan biaya tambahan ketika melewatkan data melalui HTTPS, padahal HTTPS diperlukan ketika menggunakan HTTP/2.

Komponen utama terakhir yang dibutuhkan adalah mikrokontroler. Setiap mikrokontroler membutuhkan perangkat yang mampu untuk menghubungkan mikrokontroler dengan jaringan yang akan digunakan, sehingga dalam penelitian ini digunakanlah ESP32 serta ESP8266 sebagai pengirim sekaligus penerima data dari MQTT *Broker*. Selain itu, terdapat pula DHT11 yang digunakan untuk mengukur suhu serta komponen lainnya seperti LED, *motor* maupun kipas yang dapat dikendalikan oleh mikrokontroler.

2. Perancangan Topologi Data

Topologi data berguna untuk menjelaskan alur pengiriman data yang berlangsung selama suatu proses berjalan. Topologi ini juga yang akan digunakan ketika proses pengambilan data berlangsung. Untuk lebih detailnya, bagan perancangan topologi data dapat diamati pada Gambar 3.3.



Gambar 3.3 Bagan Topologi Data

Pada sistem ini digunakan empat metode pengiriman untuk komunikasi antara *web browser* dengan *web API*. Hal ini ditunjukkan oleh Gambar 3.3 dengan notasi a, b, c dan d yang mewakili setiap skenario pengiriman data. Notasi a mewakili skenario program pengirim melalui WebSocket, notasi b untuk HTTP/1.1 SSE, notasi c untuk HTTPS sedangkan notasi d untuk HTTP/2 SSE. Untuk mempermudah penyebutan keempat skenario digunakanlah notasi abcd.

Secara keseluruhan terdapat tiga proses utama yang berlangsung selama sistem berjalan yakni pemuatan *web application* di *web browser* (**Notasi**

1a/1b/1c/1d menuju 2a/2b/2c/2d), pengiriman perintah dari *web browser* menuju mikrokontroler (**Notasi 3a/3b/3c/3d menuju 8a/8b/8c/8d**) serta pengiriman data suhu dari mikrokontroler menuju *web browser* (**Notasi 6abcd menuju 8a/8b/8c/8d**). Dari ketiga proses tersebut, pemuatan *web application* merupakan proses yang pertama kali berjalan ketika pengguna berinteraksi dengan sistem.

Proses yang pertama diawali setelah pengguna memasukkan alamat *web server* ke dalam kolom alamat *website* di suatu *web browser* guna meminta pemuatan *web application* (**Notasi 1a/1b/1c/1d**). *Web server* yang berjalan menggunakan bantuan Node.js akan membalas permintaan tersebut dengan mengirimkan dokumen HTML (**Notasi 2a/2b/2c/2d**). Dokumen tersebut selanjutnya akan disusun oleh *web browser* untuk ditampilkan. Apabila dalam penyusunan tersebut dibutuhkan dokumen tambahan seperti CSS, Javascript maupun tipe dokumen lainnya, maka web browser akan meminta dokumen tersebut menuju alamat *website* yang telah dicantumkan di dokumen HTML. Alamat *website* tidak harus menggunakan alamat *web server* yang menyediakan *web application*, seringkali beberapa dokumen maupun data dibutuhkan dari alamat lainnya semisal dari *web API* (**Notasi 3a/3b/3c/3d**). Hal inilah yang terjadi pada proses pengiriman perintah maupun permintaan data suhu oleh *web browser*.

Dalam penelitian ini, data jumlah piranti elektronik yang terhubung dengan mikrokontroler yang digunakan tidaklah tercantum di halaman HTML. Data yang tercantum di halaman tersebut hanyalah data jumlah mikrokontroler serta jumlah sensor yang digunakan. Untuk mendapatkan data jumlah piranti elektronik yang terhubung dengan mikrokontroler beserta statusnya perlu dilakukan proses

pengiriman perintah dari *web browser*. Hal ini dikarenakan data status piranti elektronik yang terhubung dengan mikrokontroler hanya didapatkan setelah perintah diterima oleh mikrokontroler (**Notasi 3a/3b/3c/3d menuju 8a/8b/8c/8d** dikerjakan). Untuk menanggulangi hal tersebut dikirimkanlah perintah kosong menuju mikrokontroler setiap kali *web application* dimuat.

Untuk memenuhi kondisi tujuan penelitian ini, proses pengiriman perintah dapat dilakukan melalui berbagai protokol sesuai dengan URL yang digunakan pada kolom alamat *website.* Protokol yang dicantumkan untuk HTTPS dan HTTP/2 adalah ‘https’, untuk HTTP/1.1 adalah ‘http’ sedangkan untuk WebSocket adalah ‘ws’. Walaupun melalui berbagai metode pengiriman data yang berbeda, pada dasarnya perintah yang dikirim berisikan data ruangan, piranti elektronik serta status piranti elektronik yang diinginkan oleh pengguna. Selanjutnya, perintah yang telah diterima oleh *web API* akan diteruskan menuju MQTT *Broker* (**Notasi 4abcd**) menggunakan susunan data yang lebih sederhana daripada susunan data yang diterima oleh *web API* (JSON). Susunan data ini dibuat supaya memudahkan pengembangan dari sisi mikrokontroler.

Setelah data diterima oleh mikrokontroler (**Notasi 5abcd**), data tersebut akan digunakan untuk menentukan perangkat mana yang statusnya akan diubah. Data status piranti elektronik tersebut kemudian disimpan di mikrokontroler untuk dikimkan kembali setelah status piranti elektronik berhasil diubah sesuai keinginan pengguna. Data yang dikirimkan menuju *web API* (**Notasi 6abcd**) tidak hanya data piranti elektronik yang berhasil diubah saja melainkan data status seluruh piranti elektronik yang berada pada mikrokontroler tersebut. Selanjutnya,

data tersebut dikirimkan menuju *web API* menggunakan format penyusunan data yang berbeda dengan susunan data yang digunakan untuk menerima data dari *web API*. Susunan dibuat dengan sesederhana mungkin untuk menanggulangi keterbasan jumlah karakter yang mampu dikirimkan dari mikrokontroler. Setelah data tersebut diterima oleh *web API* (**Notasi 7abcd**), data tersebut akan diteruskan menuju *web browser* menggunakan susunan data JSON (**Notasi 8a/8b/8c/8d**). Terakhir, data tersebut kemudian diolah oleh web browser sehingga mampu untuk dilihat serta digunakan kembali oleh pengguna.

Berbeda dengan proses pengiriman perintah, proses pengiriman suhu dilakukan dari mikrokontroler menuju *web browser* tanpa membutuhkan permintaan data terlebih dahulu (**Notasi 6abcd menuju 8a/8b/8c/8d**). Hal ini dapat terjadi dikarenakan keempat metode mampu membuka koneksi dengan rentang waktu yang cukup lama. Proses pembacaan sensor suhu yang dilakukan oleh sensor DHT11 dilakukan setiap beberapa detik. Setelah data suhu didapatkan, data suhu kemudian dikirim menuju *web API* untuk diolah (**Notasi 6abcd**) dan dikirimkan menuju *web browser* (**Notasi 8abcd**).

Dari ketiga proses tersebut, *web API* memiliki peran yang penting untuk mengolah data terutama data yang diterima dari mikrokontroler. Data yang berasal dari mikrokontroler seperti halnya suhu maupun status piranti elektronik perlu dikirimkan secara langsung dari *web API* menuju *web browser* melalui koneksi TCP yang telah terbuka (**Notasi 6abcd menuju 8a/8b/8c/8d**). Untuk memisahkan permintaan data suhu dan data status piranti elektronik (**Notasi 3a/3b/3c/3d**) agar tidak saling bercampur digunakanlah *path* tertentu pada URL yang digunakan.

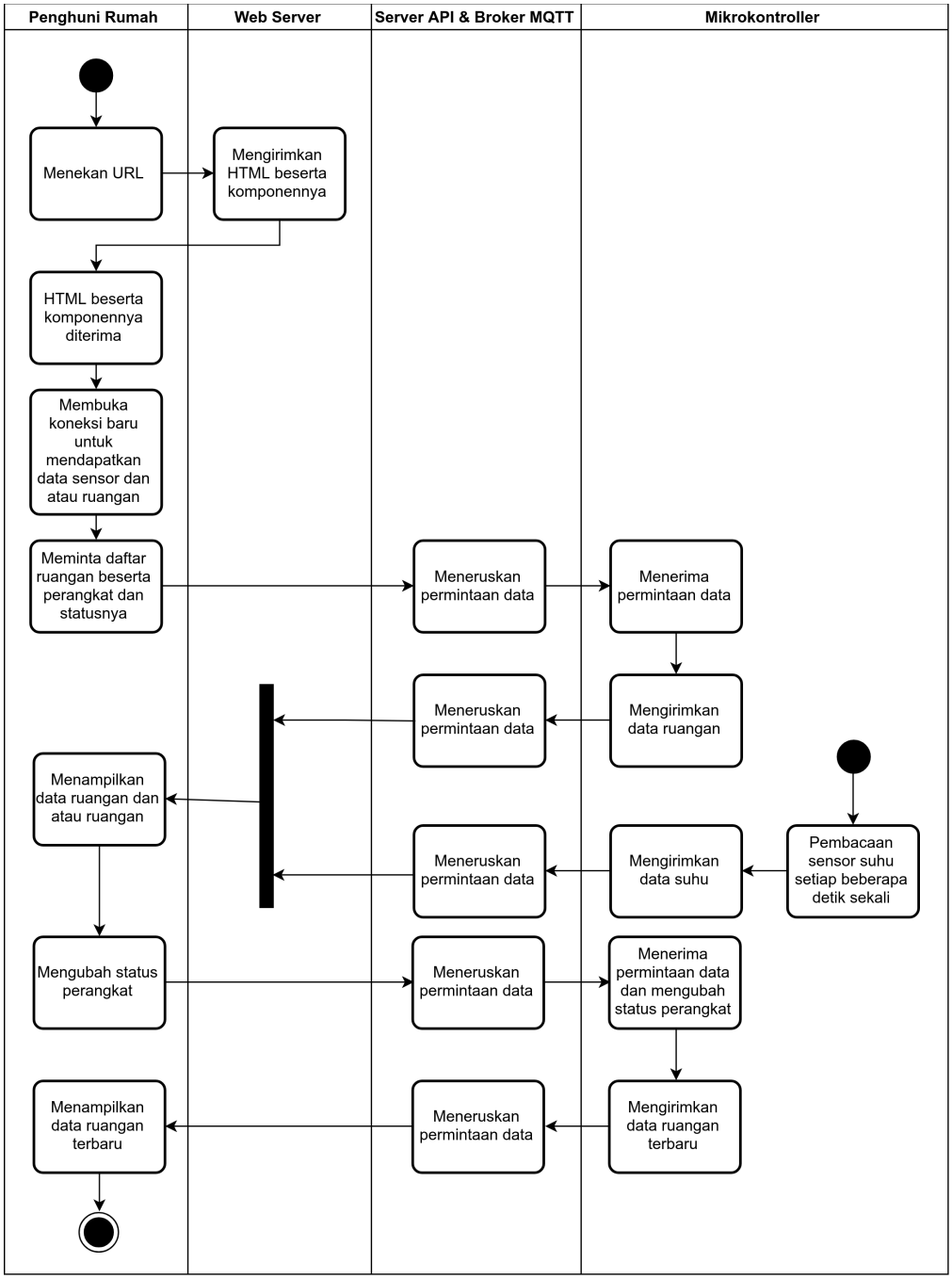
Jumlah *path* yang dapat digunakan sesuai dengan jumlah koneksi terbuka yang dibutuhkan. Koneksi terbuka yang dimaksud adalah koneksi yang memungkinkan *web API* untuk mengirimkan data menuju *web browser* selama hubungan antar keduanya masih terjalin. Pada penelitian ini dibutuhkanlah tiga koneksi terbuka yakni untuk melewatkan data status piranti elektronik, data sensor suhu dari ESP32 serta data sensor suhu dari ESP 8266. Untuk melakukan permintaan data status piranti elektronik yang terhubung pada seluruh mikrokontroler *path* yang digunakan adalah '/0', untuk permintaan data sensor suhu pada ESP32 adalah '/1' sedangkan untuk permintaan data sensor pada ESP8266 adalah '/2'.

3.4.2. Perancangan Model

Perancangan model dibuat untuk menggambarkan alur pemakaian sistem oleh pengguna. Dalam pembuatannya, perancangan model dibuat menggunakan Unified Modeling Language (UML) Diagram yang mana memudahkan pengembang maupun orang lain untuk menyampaikan alur kerja suatu sistem yang telah dibuat. UML Diagram yang digunakan dalam perancangan model adalah *use case diagram* dan *activity diagram*.

a. *Activity Diagram*

Activity Diagram digunakan untuk menggambarkan aliran kerja atau langkah-langkah yang ditempuh selama sistem berjalan. Dalam perancangan *activity diagram*, keseluruhan *activity* yang terdapat di dalam sistem ditampilkan sesuai dengan tiga komponen utama yang terdapat pada aplikasi. Pada Gambar 3.3 terlihat *activity diagram* dari sistem yang akan dibuat.

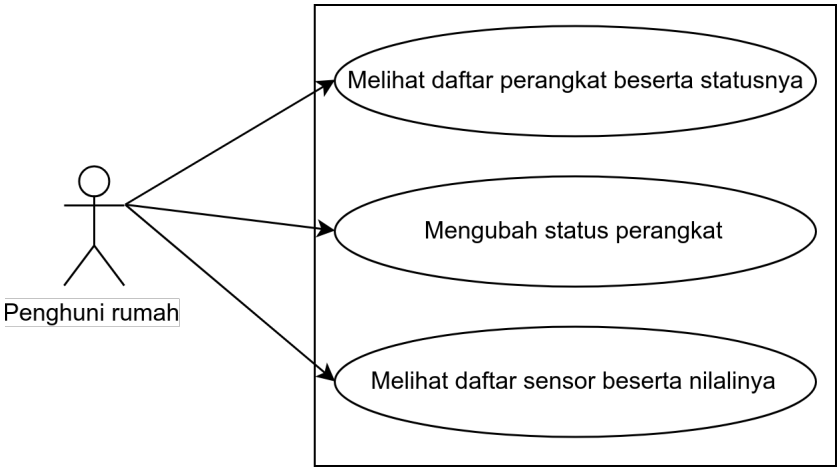


Gambar 3.3 *Activity Diagram* sistem

b. *Use Case Diagram*

Use Case Diagram adalah diagram UML yang ditujukan untuk menggambarkan skenario fungsi dari sistem yang dikembangkan. Tujuan utama

penggunaan *use case diagram* adalah untuk membuat visualisasi dari fungsi yang dibuat dalam suatu sistem. Gambar 3.4 merupakan *use case diagram* dari *web server* yang akan dibuat.



Gambar 3.4 ***Use case diagram web server***

3.5 Pembuatan *Web API* serta *Web Server*

Proses pengiriman data menggunakan metode *WebSocket* berbeda dengan ketika menggunakan *Server-Sent Events* baik di sisi *server* maupun di sisi *web browser*. Hal ini dikarenakan WebSocket mampu untuk melakukan pengiriman data dari dua arah (*bidirectional*), namun tidak dengan *Server-Sent Events.* Selain itu untuk mengirim data menggunakan protokol *WebSocket* di *Javascript* diperlukan *WebSocket API* sedangkan untuk metode *Server-Sent Events* membutuhkan *EventSource API*.

3.5.1 Penerapan Node.js serta Vue.js

Sebelum menerapkan *WebSocket* maupun *Server-Sent Events* di sisi server, perlulah dibuat proyek Node.js baru. Node.js sebagai *Javascript Framework* diperlukan untuk mempermudah pembuatan *Web API* maupun *web server.* Untuk

membuat proyek baru yang akan digunakan sebagai *Web API* dimasukkanlah perintah :

```
$ mkdir webapi && cd webapi

$ touch index.js

$ npm init
```

Setelah itu, masukkan nama beserta *metadata* npm lainnya. *Metadata* npm masih dapat diubah setelah proyek terbentuk dengan cara mengubah lalu konten yang ada pada "package.json".

Setelah *Web API* selesai dibuat, dibuatlah proyek baru yang akan digunakan sebagai *web server* digunakanlah perintah :

```
$ vue create webapp
```

Kemudian pilihlah pilihan "*Manually select features*" dan dilanjutkan dengan memilih fitur Babel dan Router. Babel merupakan *javascript compiler* yang memungkinkan berbagai *web browser* untuk menggunakan ECMAScript2015 keatas, mengubah ekstensi vue maupun jsx sehingga mampu untuk dibaca oleh web browser serta melakukan *polyfill*. Untuk fitur Router berguna untuk memudahkan pemetaan alamat (URL) pada "Vue.js".

3.5.2 Penerapan *WebSocket* dan *Server-Sent Events*

Untuk menggunakan *WebSocket API* di *web browser* dibutuhkanlah *server* yang mampu digunakan sebagai *WebSocket server.* Node.js sebagai *web API* telah mampu untuk dijadikan *WebSocket server* menggunakan *module* (*library*)

tambahan yang bernama ‘WebSocket’. Module ini dapat diunduh melalui npm setelah Web API dibuat dengan memasukkan perintah :

```
$ cd webapi

$ npm install WebSocket
```

Penggunaan *module* ‘WebSocket’ dapat dilihat pada dokumentasinya ataupun pada halaman Lampiran 2. Berbeda dengan di *Web API*, penerapan *WebSocket* di *web server* dapat menggunakan *WebSocket API* yang telah disediakan oleh HTML5, sehingga tidak diperlukan *module* tambahan.

Sama halnya dengan *WebSocket*, dalam penerapan *Server-Sent Events* pada sisi *server* dibutuhkan *module* tambahan yakni ‘express’ dan ‘http2’. Module ‘express’ berguna untuk mempermudah pembuatan *REST API* melalui HTTP/1.1 di dalam Node.js, sedangkan module ‘http2’ berguna untuk pembuatan *REST API* melalui HTTP/2. Untuk penerapan *Server-Sent Events* di dalam *web server* menggunakan *EventSource API* yang telah dimiliki oleh sebagian besar *web browser*.

3.5.3 Penerapan TLS

Tanpa menggunakan TLS, HTTP/2 tidak dapat digunakan. TLS dapat dibuat menggunakan bantuan OpenSSL. Untuk meng-*generate self-signed certificate* di dalam sistem operasi Linux Mint, dimasukkan perintah sebagai berikut :

```
$ openssl req -x509 -newkey rsa:4096 -keyout

  secret.key -out secret.crt -days 365
```

3.6 Metode Pengambilan Data

Tujuan dari penelitian ini dilakukan adalah untuk membandingan beberapa metode pengiriman data di dalam kasus rumah pintar. Metode pengiriman data yang dipilih adalah HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE serta WebSocket. Keempat metode dibandingkan berdasarkan nilai *response time* serta presentase penggunaan CPU yang didapatkan setelah proses pengambilan data dilakukan.

3.6.1 *Response Time*

Pengambilan data *response time* untuk keempat metode dilakukan dengan cara menghitung selisih waktu dari dua puluh perintah yang dikirimkan dari *web browser* sampai mendapatkan balasan dari mikrokontroler. Keduapuluh perintah dikirimkan dengan jeda waktu pengiriman yang bervariasi serta memiliki ukuran paket yang sama. Perhitungan waktu dilakukan dengan menandai waktu awal pengiriman perintah serta waktu akhir pesan balasan diterima dari sisi *web browser*. Untuk proses penandaan waktu digunakanlah *instance* Date yang dimiliki oleh Javascipt, salah satu fungsi *instance* Date adalah mengambil data waktu saat ini dari komputer pengguna dengan format awal ISO 8601.

Proses pengambilan *response time* dilakukan dalam dua skenario yang berbeda, yakni di dalam jaringan lokal yang sama serta melalui jaringan internet. Untuk kedua skenario tersebut digunakan jaringan Indihome untuk mengakses *web server* maupun *web API* dari *web browser,* namun supaya web API dapat diakses dari luar jaringan lokal dibutuhkan aplikasi pihak ketiga, yakni Serveo. Serveo adalah aplikasi yang berguna untuk melakukan *SSH Tunneling*, yang mana

memungkinkan seseorang untuk mengakses komputer yang berbeda jaringan melalui port tertentu.

3.6.2 Penggunaan CPU

Pengambilan data presentase penggunaan CPU dilakukan dengan menggunakan bantuan aplikasi ‘top’ yang merupakan aplikasi bawaan dari sistem operasi Linux Mint. Perintah yang dimasukkan untuk pengambilan data dengan menggunakan aplikasi ‘top’ adalah sebagai berikut :

```
$ top | grep <pid>
```

<pid> adalah id dari proses yang berjalan dalam sistem komputer, dalam kasus ini pid yang digunakan adalah pid dari ‘node’. Pemilihan pid ketimbang ‘node’ dikarenakan ketika aplikasi ‘node’ berjalan, terdapat beberapa aplikasi ‘node’ yang muncul di aplikasi ‘top’ sehingga dipilihlah pid dari ‘node’ dengan presentase penggunaan CPU-nya yang naik maupun turun ketika terdapat data yang melalui *web API*.

Skenario pengambilan data presentase penggunaan CPU dilakukan dengan cara mengirimkan perintah setiap detiknya dari beberapa *web browser* menuju mikrokontroler secara bersamaan. Setiap *web browser* diibaratkan sebagai satu pengguna. Setiap perintah yang dikirimkan dari *web browser* akan dibalas oleh mikrokontroler. Perintah yang dikirimkan dari *web browser* memiliki ukuran paket yang sama, begitu juga dengan perintah yang dikirimkan dari mikrokontroler. Selain menerima balasan dari mikrokontroler, *web browser* juga menerima dua data suhu yang dikirimkan melalui *stream* yang berbeda antar keduanya maupun dengan *stream* yang digunakan untuk menerima pesan dari

mikrokontroler. Untuk jumlah *web browser* yang digunakan akan bervariasi selama proses pengambilan data.

Dua w*eb browser* yang sama ketika dibuka pada waktu yang bersamaan akan menggunakan *session*, *history*, maupun *cache* yang sama. Hal ini tidak mengganggu proses pengambilan data WebSocket maupun HTTP/2 SSE yang memiliki karakteristik paralel, namun tidak dengan HTTP/1.1 SSE maupun HTTPS SSE yang hanya mampu membuka enam TCP *Connection.* Untuk mengatasi hal tersebut dibuat *session* yang berbeda untuk setiap membuka *web browser* dengan menjalankan *script* berikut di terminal Linux.

```
#!/bin/bash
RND_DIR="chromee-$RANDOM"
echo $RND_DIR
cd /tmp
mkdir $RND_DIR
google-chrome --user-data-dir=/tmp/$RND_DIR
--incognito
rm -R $RND_DIR
```

BAB IV

HASIL PENELITIAN DAN PEMBAHASAN

Pada bab ini akan menjelaskan mengenai hasil perbandingan antara HTTP/1.1 *Server-Sent Events*, HTTP/2 *Server-Sent Events*, HTTPS *Server-Sent Events* serta dengan WebSocket sebagai program pengirim yang digunakan pada sistem kendali rumah pintar. Proses pengambilan data dari keempat program pengirim yang digunakan pada penelitian ini menggunakan jaringan internet Indihome pada waktu siang hari. Untuk parameter yang dibandingkan berupa *response time* serta presentase penggunaan CPU.

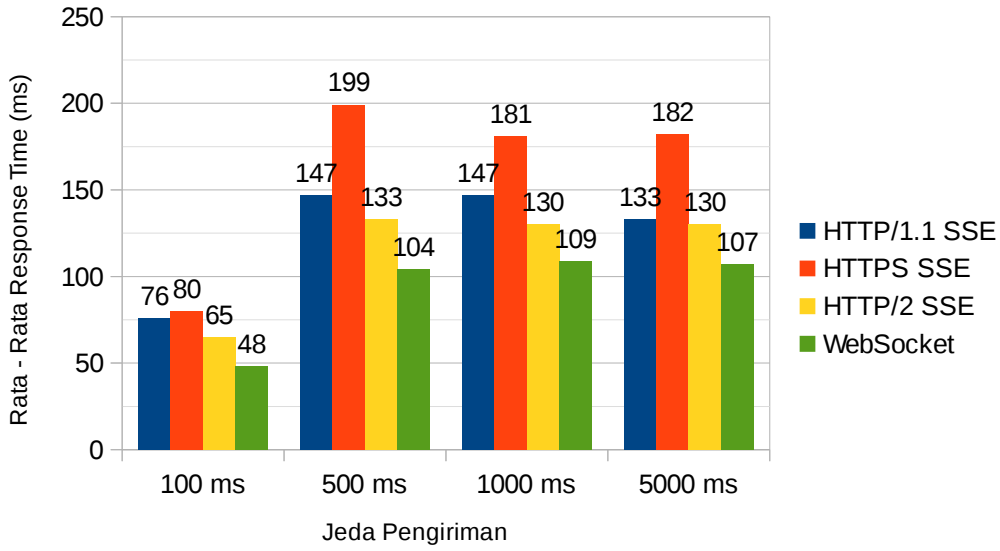
4.1 Hasil Perbandingan *Response Time*

Response time merupakan lama waktu yang dibutuhkan oleh *web browser* untuk mendapatkan balasan dari mikrokontroler setelah *web browser* melakukan pengiriman perintah menuju mikrokontroler. Hal ini ditunjukkan Gambar 3.3 dengan notasi dari 3a/3b/3c/3d menuju 8a/8b/8c/8d. Data yang didapatkan oleh *web browser* berupa status piranti elektronik yang terhubung dengan mikrokontroler. Dalam perjalanannya, data – data tersebut melewati MQTT *Broker* serta *web API. Web API* tidak harus berada di dalam jaringan lokal yang sama sengan *web browser*. Semakin jauh jarak tempuh data dari *web browser* menuju *web API*, semakin besar pula nilai *response time* nya. Dengan bertambahnya nilai *response time*, tingkat variasi data yang didapatkan juga semakin beragam. Oleh karena itu, dibuatlah dua skenario pengambilan data *response time* yakni ketika *web browser* berada di dalam jaringan lokal yang sama

dengan *web API* maupun ketika *web browser* terhubung dengan *web API* melalui jaringan internet.

4.1.1 Skenario Jaringan Lokal

Pada skenario ini dilakukan pengiriman lima puluh pesan dari *web browser* menuju mikrokontroler dengan kondisi *web browser* berada di dalam jaringan lokal yang sama dengan *web API*. Hasil yang didapatkan dari pengiriman kelimapuluh pesan telah dirangkum pada Gambar 4.1.

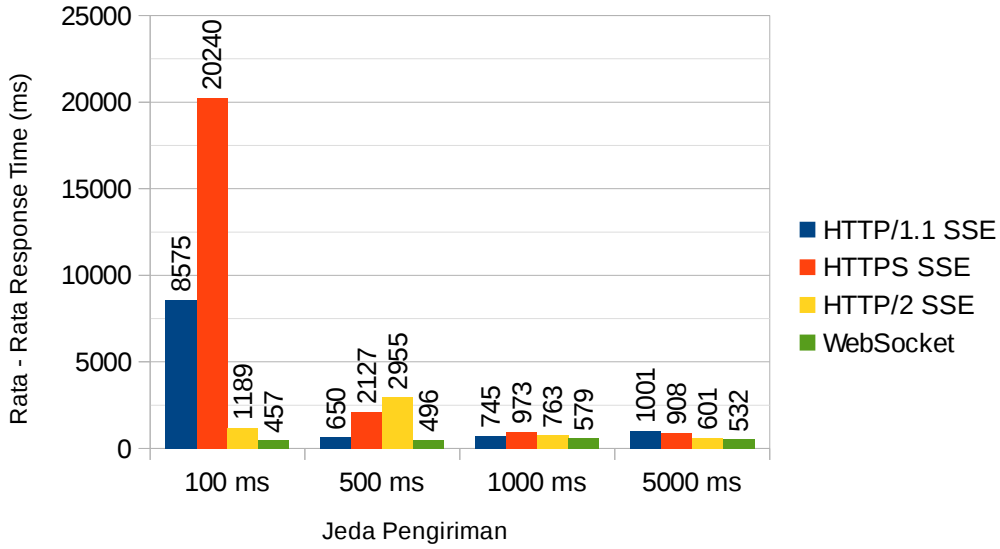


Gambar 4.1 Grafik rata - rata *response time* pada skenario jaringan lokal

Berdasarkan hasil yang didapatkan pada percobaan ini dapat dinyatakan bahwa WebSocket memiliki nilai rata - rata *response time* terkecil. Selain itu, nilai rata-rata *response time* dari keempat metode selalu memiliki urutan yang sama. Urutan dapat dimulai dari pemilik rata - rata *response time* terkecil yakni WebSocket, HTTP/2 SSE, HTTP/1.1 SSE sampai dengan HTTPS SSE.

4.1.1 Skenario Jaringan Internet

Proses pengambilan data pada skenario ini menggunakan cara yang sama dengan skenario jaringan lokal, namun dengan kondisi *web browser* terhubung dengan *web API* melalui jaringan internet dengan bantuan *tunneling* oleh Serveo. Hasil yang didapatkan dari percobaan ini telah dirangkum pada Gambar 4.2.



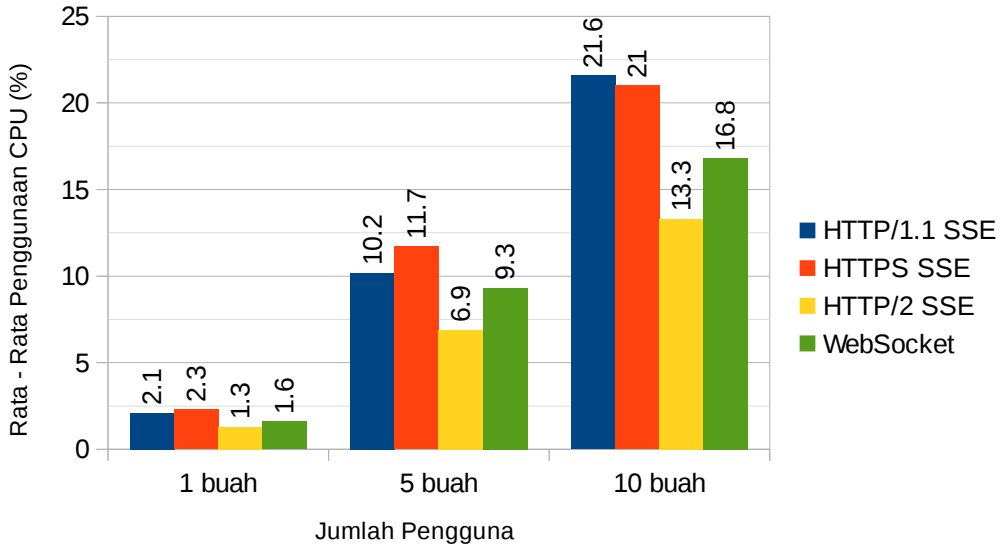
Gambar 4.2 Grafik rata - rata response time pada skenario jaringan internet

Variasi data yang didapatkan ketika melalui jaringan internet jauh berbeda dibandingkan dengan skenario jaringan lokal, namun untuk hasil yang didapatkan tetaplah sama yakni WebSocket memiliki rata - rata response time terkecil dibandingkan dengan ketiga metode lainnya. Hasil lain yang didapatkan dari percobaan ini adalah ketika digunakannya Serveo ada **kemungkin hilangnya suatu pesan yang dikirimkan oleh mikrokontroler maupun *web browser*** ketika pengiriman dilakukan melalui HTTP/1.1 maupun HTTP/1.1 *over* TLS. Selain itu ketika data dikirimkan melalui HTTP/1.1 atau HTTP/1.1 *over* TLS dengan jeda pengiriman lima detik didapati bahwa Serveo berhenti bekerja pada

urutan pengiriman tertentu. Untuk lebih detail hasil percobaan dapat dilihat pada Lampiran 3.

4.2 Hasil Perbandingan Penggunaan CPU

Proses pengambilan data presentase penggunaan CPU dilakukan dengan mengirimkan perintah menuju mikrokontroler dengan rentang satu detik selama percobaan berlangsung. Selama itu pula dilakukan pengamatan terhadap kenaikan presentase penggunaan CPU oleh Node.js pada perangkat *server* Raspberry Pi. Hasil yang didapatkan dari percobaan ini telah dirangkum pada Gambar 4.3.



Gambar 4.3 Grafik rata - rata penggunaan CPU

Berdasarkan hasil yang diperoleh pada percobaan ini dapat dinyatakan bahwa HTTP/2 SSE memiliki nilai rata - rata presentase penggunaan CPU terkecil. Selain itu, nilai rata – rata presentase penggunaan CPU dari keempat metode cenderung memiliki urutan yang sama. Urutan dapat dimulai dari pemilik rata - rata presentase penggunaan CPU terkecil yakni HTTP/2 SSE, WebSocket, HTTP/1.1 SSE sampai dengan HTTPS SSE.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan data yang diperoleh serta analisis yang sudah dilakukan, maka dapat ditarik beberapa kesimpulan dari penelitian tersebut, diantaranya :

1. WebSocket memiliki nilai *response time* terkecil di antara keempat metode lainnya.
2. HTTP/2 *Server-Sent Events* memiliki nilai presentase penggunaan CPU terkecil di antara keempat metode lainnya.
3. HTTPS *Server-Sent Events* cenderung memiliki nilai terbesar baik dalam pengujian *response time* maupun presentase penggunaan CPU.
4. Terdapat kemungkinan paket yang hilang selama penggunaan Serveo untuk HTTP/1.1 *Server-Sent Events* maupun HTTPS *Server-Sent Events*.

5.2 Saran

Pada pengembangan selanjutnya untuk memperbaiki proyek akhir ini dapat dilakukan beberapa pertimbangan yaitu :

1. Untuk mendapatkan data *response time* yang lebih akurat, dapat dilakukan beberapa pengujian *response time* pada saat yang berbeda-beda seperti halnya ketika pagi, siang, sore dan malam.
2. Menambahkan parameter pengujian *delay* atau selisih waktu dari sisi mikrokontroler maupun *web API* menuju *web browser.*

LAMPIRAN

LAMPIRAN 1 – Instalasi

1. Instalasi Mosquitto

Instalasi Mosquitto dapat dilakukan dengan cara mengunduh langsung melalui *default repository* Raspberry Pi 3.

```
$ sudo apt update

$ sudo apt install -y mosquitto mosquitto-clients
```

Pada penginstalan di atas, “mosquitto” adalah nama paket *broker* untuk protokol MQTT sedangkan “mosquitto-clients” adalah paket pendukung untuk melakukan percobaan penggunaan protokol MQTT sebagai *client*. Setelah instalasi dilakukan, cek status Mosquitto dengan perintah:

```
$ service mosquitto status
```

Untuk menjalankan Mosquitto secara otomatis setelah Raspberry Pi 3 dihidupkan adalah dengan memasukkan perintah berikut.

```
$ sudo systemctl enable mosquitto.service
```

Setelah Mosquitto aktif, lakukan percobaan menggunakan mosquitto-client. Bukalah terminal baru dan masukkan perintah berikut untuk menjalankan *client* sebagai *subscriber* :

```
$ mosquitto_sub -h localhost -t testing
```

Kemudian buka terminal baru lagi dan masukkan perintah berikut untuk menjalankan *client* sebagai *publisher* :

```
$ mosquitto_pub -h localhost -t testing -m hai
```

Pada kedua perintah di atas, “testing” adalah *topic* yang digunakan untuk oleh protokol MQTT sedangkan “hai” adalah pesan yang dikirimkan oleh *publisher* yang akan diterima oleh *subscriber* dengan *topic* yang sama.

2. Instalasi Node.js

Sebelum menginstall Node.js, periksa terlebih dahulu *processor* yang digunakan oleh Raspberry Pi dengan perintah :

```
$ uname -m
```

Sebagai catatan, seluruh Raspberry Pi 3 memiliki processor dengan model ARMv8, sehingga pada penelitian ini diunduh Node.js untuk Linux Binaries (ARM) dengan opsi ARMv8 dari halaman resminya (https://nodejs.org/en/download/). Setelah Node.js berhasil diunduh, masuk pada folder Downloads dan ekstraklah hasil unduhan dengan perintah :

```
$ cd ~/Downloads

$ tar -xzf node-v10.15.3-linux-arm64.tar.xz
```

Kemudian salinlah hasil ekstrak Node.js ke *directory* /usr/local/

```
$ cd node-v10.15.3-linux-arm64

$ sudo cp -R * /usr/local/
```

Setelah itu, untuk cek apakah instalasi berhasil dengan cara mengecek versi npm dan node yang telah di-*install*,

```
$ npm -v

$ node -v
```

Setelah node dan npm berhasil ter-*install*, *install* modul Vuejs-cli dengan memasukkan perintah:

```
$ npm install -g @vue-cli
```

LAMPIRAN 2 – Source Code

A. *Source Code* untuk ***Web Browser***

1. *Source Code* untuk **keempat program pengirim**

   a. Untuk menerima data dari *Web API*

```
client.addEventListener("message",e =>
{ console.log(e) //e adalah pesan yang diterima
})
```

   b. Apabila koneksi antara *web browser* dengan *web API* telah terhubung maka akan menjalankan perintah tertentu.

```
client.onopen = () => {
console.log("koneksi telah terjalin")
}
```

2. *Source Code* untuk **WebSocket** pada sisi *web browser*

   a. Untuk membuka koneksi dengan *Web API*

```
client = new WebSocket(

"ws://<Web API Domain / IP Address>" + "/" + <path>);
```

   b. Untuk mengirim data menuju *Web API*

```
client.send(<message>);
```

3. Source Code untuk **HTTP/1.1 SSE** pada sisi *web browser*

   a. Untuk membuka koneksi dengan *Web API*

```
client = new EventSource(
"http://<Web API Domain / IP Address>" + "/" +
```

```
<path>);
```

b. Untuk mengirim data menuju *Web API*

```
fetch("https://<Web API Domain / IP Address>", {
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    body: <message>
}).then(response => {
    console.log(response);
});
```

4. Source Code untuk **HTTPS SSE** pada sisi *web browser*

a. Untuk membuka koneksi dengan *Web API*

```
client = new EventSource(
"https://<Web API Domain / IP Address>" + "/" +
<path>);
```

b. Untuk mengirim data menuju *Web API*

```
fetch("https://<Web API Domain / IP Address>", {
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    body: <message>
}).then(response => {
    console.log(response);
});
```

5. Source Code untuk **HTTP/2 SSE** pada sisi *web browser*

a. Untuk membuka koneksi dengan *Web API*

```
client = new EventSource(
"https://<Web API Domain / IP Address>" + "/" +
<path>);
```

b. Untuk mengirim data menuju *Web API*

```
fetch("https://<Web API Domain / IP Address>", {
```

```
    method: "POST",
    body: <message>
}).then(response => {
    console.log(response);
});
```

B. *Source Code* untuk ***Web API***

1. *Source Code* untuk **MQTT**

a. Untuk membuka koneksi antara Web API dengan MQTT Broker

```
const mqtt = require('mqtt')
var client = mqtt.connect(
'mqtt://<MQTT Broker Domain / IP Address>')
```

b. Untuk menerima data dari MQTT Broker

```
client.subscribe(<topic>);
client.on('message', function (topic, message) {
     // message dalam bentuk buffer
     console.log(message.toString('utf-8'))
});
```

c. Untuk mengirim data menuju MQTT Broker

```
client.publish(<topic>, <message>)
```

2. *Source Code* untuk **WebSocket**

a. Membuat *Server API*

```
var WebSocketServer = require('websocket').server;
var http = require('http');

var server = http.createServer(function (request,
response) {
    console.log(request.headers);
    response.writeHead(404);
    response.end();
});

server.listen(1337, function ()
{ console.log(`websocket on port 1337!`) });
```

```
var wsServer = new WebSocketServer({
    httpServer: server
});
```

b. Menangkap permintaan data dari *web browser*

```
wsServer.on('request', function (request) {

    var connection = request.accept()

    //mengirim pesan menuju web browser
    connection.sendUTF(<message>);

    //menerima pesan dari web browser
    connection.on('message', function (message) {
        console.log(JSON.parse(message.utf8Data))
    });

    //ketika koneksi antara web browser
    //dengan web API terputus
    connection.on('close', function (reasonCode,
description) {
        console.log(connection.remoteAddress + '
disconnected.');
    });
});
```

3. *Source Code* untuk **HTTP/1.1 SSE**

a. Membuat *Server API*

```
const express = require('express')
var cors = require('cors');
const app = express()

app.use(cors())
app.use(express.static('public'))
app.use(express.json());

app.listen(3000, function () { console.log(`http/1.1
on port 3000!`) });
```

b. Menangkap permintaan data dari *web browser*

```
app.get('/:index', (req, res) => {
     res.writeHead(200, {
```

```
          'Content-Type': 'text/event-stream',
          'Cache-Control': 'no-cache',
          'Connection': 'keep-alive',
     });
     res.write('\n');
     res.write(
     "event: message\ndata: " +<message>+ "\n\n")
});

app.post('/', (req, res) => {
     console.log(req.body)
     res.send('OK');
});
```

4. *Source Code* untuk **HTTPS SSE**

a. Membuat *Server API*

```
const express = require('express')
var cors = require('cors');
const app = express()
var https = require('https')

app.use(cors())
app.use(express.static('public'))
app.use(express.json());

https.createServer({
    key: fs.readFileSync('./cert/server.key'),
    cert: fs.readFileSync('./cert/server.crt')
}, app)
  .listen(3002, function () {
    console.log(`https 1.1 on port 3002!`)
})
```

b. Menangkap permintaan data dari *web browser*

```
app.get('/:index', (req, res) => {
     res.writeHead(200, {
          'Content-Type': 'text/event-stream',
          'Cache-Control': 'no-cache',
          'Connection': 'keep-alive',
     });
     res.write('\n');
     res.write(
     "event: message\ndata: " +<message>+ "\n\n")
```

```
});

app.post('/', (req, res) => {
    console.log(req.body)
     res.send('OK');
});
```

4. *Source Code* untuk **HTTP/2 SSE**

a. Membuat *Server API*

```
const express = require('express')
var cors = require('cors');
const app = express()
var https = require('https')

app.use(cors())
app.use(express.static('public'))
app.use(express.json());

https.createServer({
    key: fs.readFileSync('./cert/server.key'),
    cert: fs.readFileSync('./cert/server.crt')
}, app)
  .listen(3002, function () {
    console.log(`https 1.1 on port 3002!`)
})
```

b. Menangkap permintaan data dari *web browser*

```
const http2 = require('http2');
http2.createSecureServer({
     key: fs.readFileSync('./cert/server.key'),
     cert: fs.readFileSync('./cert/server.crt')
},
(req, res) => {
     if (req.headers[':method'] == 'POST') {
          let body = '';
          req.setEncoding('utf8');
          req.on('data', (chunk) => {
               body += chunk
          })
          req.on('end', () => {
               try{
                //menerima pesan dari web browser
                    console.log(body);
```

```
                    res.end()
                }catch(err){
                    res.statusCode = 400;
                    res.end(err)
                }
            })
            res.setHeader(
           'Access-Control-Allow-Origin', '*')
            res.write('OK')

     } else
    if (req.headers[':method'] == 'GET') {
     new Promise((resolve) => {
          res.statusCode = 200;
          res.setHeader(
         'Content-Type', 'text/event-stream');
          res.setHeader(
         'Cache-Control', 'no-cache')
          res.setHeader(
         'Access-Control-Allow-Origin', '*')
          res.write(`\n`)
          resolve(null)
     }).then(() => {
                //mengirim pesan menuju web browser
                    res.write(
                "event: message\ndata: "+msg+"\n\n")
                res.end()
            })
     })}
}).listen(
     3001, () => {
          console.log("http/2 on port 3001");
     }
)
```

LAMPIRAN 3 – Hasil Penelitian

A. Pengujian *response time* pada jaringan lokal

1. HTTP/1.1 SSE

	Response Time (ms)			
No	Jeda Pengiriman 100 ms	Jeda Pengiriman 500 ms	Jeda Pengiriman 1000 ms	Jeda Pengiriman 5000 ms
1	154	166	78	94
2	68	96	105	120
3	25	95	83	130
4	50	104	48	44
5	53	115	75	61
6	68	215	99	82
7	79	230	119	96
8	34	49	236	118
9	60	58	84	234
10	72	71	92	47
11	24	86	116	65
12	79	107	230	85
13	35	111	63	100
14	205	83	85	121
15	118	222	82	138
16	35	234	223	81
17	93	53	53	90
18	51	71	1397	91
19	91	93	411	105
20	31	113	218	216

21	91	228	46	243
22	29	58	71	54
23	23	87	96	93
24	111	114	210	92
25	30	132	144	311
26	36	160	64	840
27	114	80	94	146
28	33	105	112	58
29	24	215	229	87
30	125	48	61	101
31	30	71	79	110
32	30	95	107	223
33	131	210	87	145
34	46	1257	50	61
35	126	267	72	79
36	246	90	210	100
37	160	115	121	114
38	158	144	45	226
39	66	58	93	165
40	32	85	92	59
41	159	109	119	82
42	78	221	234	103
43	21	91	62	120
44	43	262	179	230
45	151	104	103	256
46	83	213	227	81
47	25	44	54	89

48	51	77	82	106
49	52	97	106	128
50	57	117	220	38
Rata-rata	75.72	146.52	147.32	133.16

2. HTTP/2 SSE

	Response Time (ms)			
No	Jeda Pengiriman 100 ms	Jeda Pengiriman 500 ms	Jeda Pengiriman 1000 ms	Jeda Pengiriman 5000 ms
1	65	110	194	112
2	57	123	72	229
3	58	210	100	45
4	61	143	211	63
5	35	159	234	93
6	70	94	83	292
7	74	78	180	115
8	110	96	122	221
9	71	104	229	239
10	30	126	72	172
11	83	128	86	100
12	33	223	111	103
13	85	229	222	119
14	36	68	64	240
15	93	73	83	55
16	30	90	104	73
17	24	97	216	90
18	93	111	48	193

19	24	123	71	125
20	120	212	98	229
21	32	223	122	62
22	109	237	234	78
23	29	65	67	93
24	110	77	92	110
25	29	95	116	216
26	31	104	226	233
27	121	117	61	59
28	30	129	275	82
29	126	240	232	96
30	37	228	324	202
31	126	110	65	220
32	37	77	79	51
33	27	101	94	68
34	131	173	215	84
35	40	108	47	102
36	33	123	78	116
37	141	132	94	223
38	56	221	121	50
39	28	235	232	69
40	145	117	64	88
41	61	75	89	105
42	27	89	42	209
43	51	102	225	226
44	54	114	60	53
45	47	126	84	71
46	59	216	107	91
47	60	228	218	106

48	62	59	56	123
49	66	70	77	230
50	71	84	101	90
Rata-rata	64.56	133.44	129.94	130.28

3. HTTPS SSE

	Response Time (ms)			
No	Jeda Pengiriman 100 ms	Jeda Pengiriman 500 ms	Jeda Pengiriman 1000 ms	Jeda Pengiriman 5000 ms
1	394	183	111	185
2	306	244	228	101
3	137	61	72	117
4	283	75	83	222
5	159	89	109	1264
6	123	114	217	71
7	41	193	156	85
8	80	123	78	106
9	21	136	108	119
10	87	227	1764	225
11	24	59	782	178
12	29	171	75	75
13	95	91	106	89
14	24	104	205	108
15	28	105	230	135
16	105	197	253	229
17	29	209	121	165
18	29	110	199	80

19	109	234	134	194
20	31	52	55	112
21	114	83	89	136
22	43	86	108	232
23	116	99	218	165
24	39	111	49	81
25	27	123	72	98
26	123	225	100	200
27	34	230	211	134
28	24	175	233	235
29	128	70	67	168
30	45	109	94	86
31	32	93	131	98
32	139	103	228	120
33	52	196	60	139
34	25	118	88	50
35	40	220	112	289
36	44	232	221	111
37	47	59	160	1317
38	48	88	81	208
39	52	93	115	138
40	52	106	218	53
41	58	199	239	72
42	69	1591	74	88
43	61	1104	97	106
44	65	620	121	210
45	66	479	232	148
46	73	145	71	57
47	23	79	92	73

48	77	117	202	95
49	56	114	141	109
50	82	122	59	215
Rata-rata	79.76	199.32	181.38	181.82

4. WebSocket

	Response Time (ms)			
No	Jeda Pengirim 100 ms	Jeda Pengirim 500 ms	Jeda Pengirim 1000 ms	Jeda Pengirim 5000 ms
1	43	123	76	57
2	51	31	192	91
3	48	42	217	95
4	50	54	42	206
5	46	83	66	223
6	57	81	93	39
7	59	95	209	57
8	61	105	35	91
9	16	206	66	95
10	66	218	179	206
11	17	36	201	223
12	71	74	30	43
13	12	56	60	63
14	76	76	108	81
15	13	87	104	99
16	79	107	220	209
17	81	222	88	29
18	11	212	74	46

19	87	30	103	67
20	11	42	215	84
21	93	57	39	201
22	13	67	63	213
23	17	80	93	32
24	111	92	208	53
25	45	193	53	70
26	23	205	57	88
27	108	218	84	105
28	24	38	201	217
29	18	49	27	36
30	114	59	44	58
31	22	73	273	75
32	17	85	106	92
33	120	99	219	202
34	33	224	47	220
35	22	211	74	52
36	127	30	95	83
37	30	68	212	79
38	29	55	41	105
39	30	66	63	205
40	49	91	90	223
41	35	182	207	43
42	38	155	32	59
43	40	205	60	81
44	44	218	81	98
45	54	36	109	209
46	56	51	25	28
47	58	58	55	45

48	22	74	79	66
49	62	85	100	84
50	11	97	217	101
Rata-rata	48.4	104.02	108.64	106.54

B. Pengujian *response time* pada jaringan internet

1. HTTP/1.1 SSE

	Response Time (ms)			
No	Jeda Pengirim 100 ms	Jeda Pengirim 500 ms	Jeda Pengirim 1000 ms	Jeda Pengirim 5000 ms
1			2004	2410
2	4914	1955	826	504
3	5034	1496	1849	1018
4	3754	1075	641	644
5	4129	1592	562	839
6	4047	1079	488	662
7	4489	600	510	1083
8	7156	591	635	595
9	7074	503	1478	926
10	5587	447		531
11	5505	526	708	1078
12	5422	537	644	594
13	5577	556	548	1196
14	5488	459	470	486
15	5998	515	487	2987
16	7911	1101	1133	469
17	8300	616	746	

18	6417	626	454	
19	6817	518	496	
20	6729	531	515	
21	7126	542	639	
22	9253	446	1081	
23	9534	466	688	
24	7440	479	611	
25	7852	491	1144	
26	7768	502	762	
27	8266	927	468	
28	10582	447	1111	
29	8673	533	627	
30	8592	549	551	
31	8990	461	470	
32	8903	470	493	
33	9297	481	518	
34	9211	447	1360	
35	12417	506	971	
36	12535	520	588	
37	10120	533	510	
38	10031	946	739	
39	10330	457	446	
40	10248	670	1094	
41	13655	476	712	
42	10564	513	631	
43	14048	500	553	
44	10963	494	476	
45	11370	526	518	
46	11280	448	1238	

47	11675	566	853	
48	14989	450	459	
49	12093	985	486	
50	12004		519	
Rata-rata	8574.63	649.04	745.1	1001.375

2. HTTP/2 SSE

	Response Time (ms)			
No	Jeda Pengiman 100 ms	Jeda Pengiman 500 ms	Jeda Pengiman 1000 ms	Jeda Pengiman 5000 ms
1	556	549	527	873
2	468	1961	584	499
3	442	1471	519	603
4	467	979	542	564
5	489	763	548	739
6	451	490	488	464
7	464	510	1412	483
8	463	509	723	504
9	471	522	458	1938
10	450	534	769	537
11	472	545	506	742
12	474	557	528	466
13	485	467	837	489
14	478	514	512	506
15	485	538	503	525
16	435	1517	706	542
17	491	1024	2011	563

18	438	817	2396	478
19	496	544	1402	1007
20	436	575	512	598
21	501	897	581	531
22	496	604	467	732
23	438	10810	599	456
24	509	10460	3013	505
25	440	9979	2020	682
26	501	9485	1027	598
27	439	8993	667	819
28	538	8499	586	549
29	453	8010	547	549
30	523	7518	636	482
31	492	7024	664	689
32	3174	6530	597	602
33	3091	6037	526	561
34	3003	5541	730	738
35	2909	5046	464	463
36	2816	4551	492	482
37	2729	4054	632	510
38	2639	3562	623	521
39	2545	3068	557	533
40	2451	2572	568	560
41	2359	2076	593	467
42	2265	1642	535	487
43	2170	1147	627	504
44	2075	1017	479	525
45	1982	745	500	728
46	1887	561	523	560

47	1793	476	636	469
48	1698	483	656	578
49	1602	487	583	513
50	1507	502	517	529
Rata-rata	1188.72	2955.24	762.56	600.84

3. HTTPS SSE

	Response Time (ms)			
No	Jeda Pengiriman 100 ms	Jeda Pengiriman 500 ms	Jeda Pengiriman 1000 ms	Jeda Pengiriman 5000 ms
1				3208
2	16893	5056	3898	543
3	16942	4172	1509	966
4	16054	2184	3056	578
5	16497	2286	1533	1020
6	16425	530	1662	612
7	16865	7400	1276	1139
8	16783	6412	6290	464
9	17187	5117	889	970
10	17105	5636	542	581
11	17486	3231	545	1008
12	17405	3746	465	615
13	18102	3345	491	1068
14	18022	5856	1026	461
15	18018	2959	465	841
16	19934	3471		512
17	20229	3083	485	1011

18	18340	2593	508	820
19	18748	2190	559	959
20	18666	2701	1050	464
21	19043	1803	668	996
22	18957	2321	587	583
23	19364	1917	536	1047
24	19283	2647	736	781
25	19668	1158	471	944
26	19583	1260	991	480
27	22984	2771	516	981
28	22906	1376	550	3675
29	20399	2887	550	541
30	20307	987	489	982
31	20609	1496	514	556
32	20522	1106	969	1003
33	20920	614	465	592
34	20831	628	483	1038
35	21124	544	515	626
36	21038	558	534	1072
37	21343	570	742	476
38	21260	475	971	1013
39	25144	1001	505	511
40	25058	924	528	939
41	22175	493	554	547
42	22091	520	471	
43	22468	536	1911	
44	22381	552	1218	
45	23081	564	833	
46	22995	462	473	

47	23008	515	490	
48	22929	504	512	
49	27328	518	722	
50	27240	528	971	
Rata-rata	20239.59	2126.59	973.4167	907.88

4. WebSocket

	Response Time (ms)			
No	Jeda Pengiriman 100 ms	Jeda Pengiriman 500 ms	Jeda Pengiriman 1000 ms	Jeda Pengiriman 5000 ms
1	556	503	595	585
2	470	503	518	1935
3	412	504	439	621
4	426	507	483	535
5	432	449	481	452
6	536	514	819	472
7	457	525	437	478
8	435	503	455	403
9	439	503	685	420
10	409	462	607	642
11	445	502	529	456
12	446	504	758	466
13	449	504	474	491
14	454	510	498	405
15	454	520	419	414
16	456	427	438	435
17	458	503	528	458
18	463	503	520	470

19	413	504	824	801
20	466	506	426	422
21	409	503	461	529
22	473	501	477	448
23	412	513	817	483
24	480	525	420	685
25	410	501	458	491
26	495	433	473	392
27	412	502	504	425
28	412	502	424	448
29	491	452	441	567
30	416	515	465	506
31	422	522	491	603
32	541	526	407	415
33	460	432	437	424
34	519	504	459	471
35	451	503	484	572
36	531	502	1002	665
37	458	504	948	607
38	516	504	868	522
39	433	508	1815	443
40	436	513	830	438
41	544	455	423	473
42	495	502	453	482
43	435	446	682	813
44	454	502	603	446
45	529	502	526	441
46	441	505	448	452
47	412	502	466	475

48	432	505	495	492
49	417	517	512	513
50	413	428	747	530
Rata-rata	456.5	495.7	579.38	532.24

C. Pengujian presentase penggunaan CPU

1. HTTP/1.1 SSE

No	Presentase penggunaan CPU (%)		
	1 pengguna	5 pengguna	10 pengguna
1	2	10.9	26.2
2	2.3	11.9	23.8
3	2.3	10.2	21.5
4	2.6	10.8	22.8
5	2.6	12.3	25.2
Rata-rata	2.133	10.183	21.583

2. HTTP/2 SSE

No	Presentase penggunaan CPU (%)		
	1 pengguna	5 pengguna	10 pengguna
1	1.3	6.2	14.4
2	1.3	8.3	12.9
3	1.6	6.6	14.8
4	1.3	8.9	12.5
5	1.3	6.5	15.1
Rata-rata	1.3	6.9167	13.283

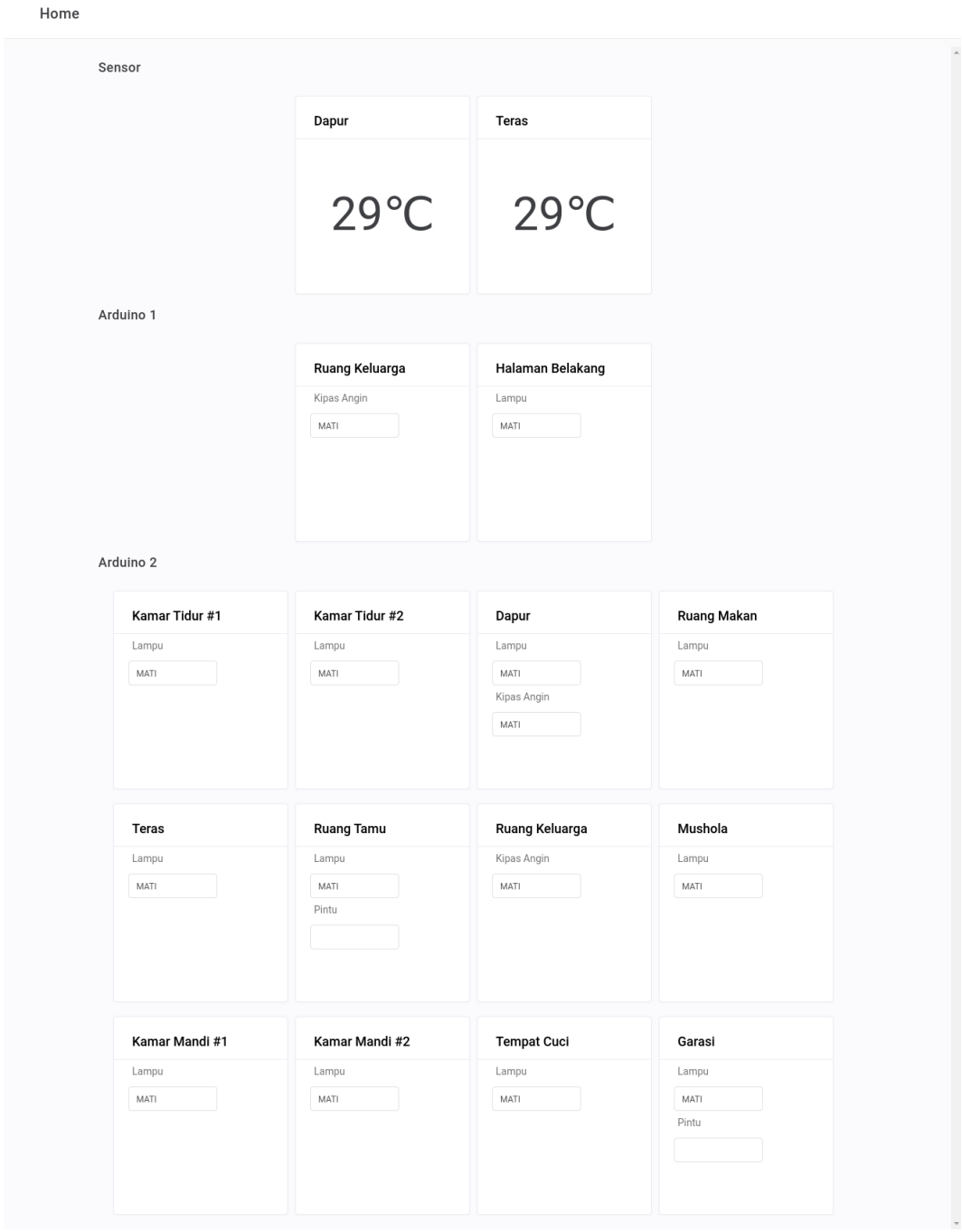
3. HTTPS SSE

No	Presentase penggunaan CPU (%)		
	1 pengguna	5 pengguna	10 pengguna
1	2.6	14.4	21.8
2	2	12.8	22.8
3	3	12.9	24.1
4	2	11.8	22.2
5	3	13.4	25.1
Rata-rata	2.2667	11.7167	21

4. WebSocket

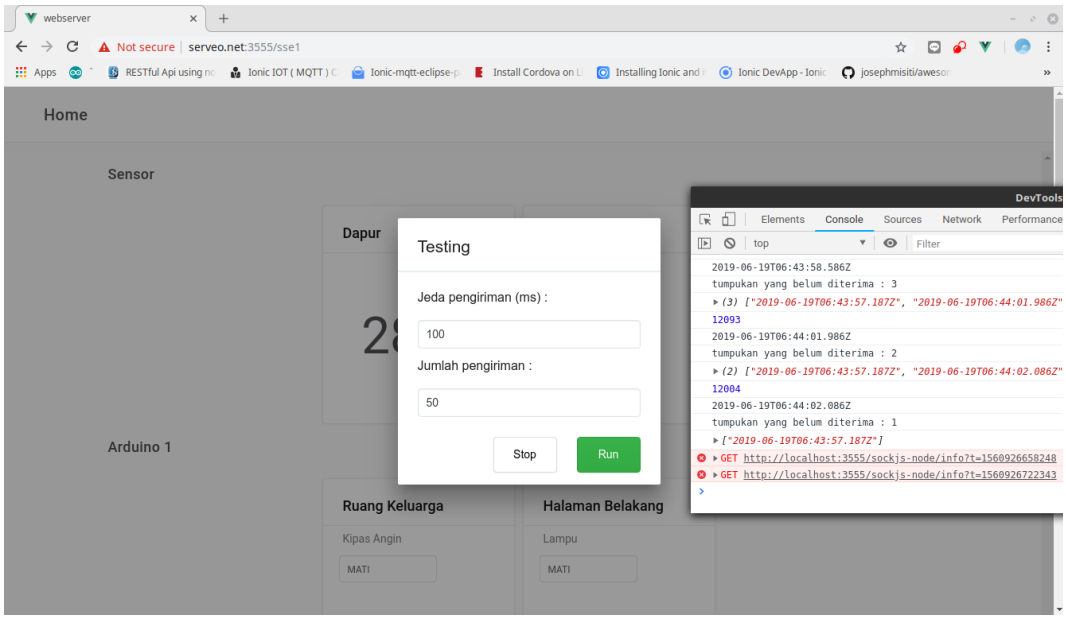
No	Presentase penggunaan CPU (%)		
	1 pengguna	5 pengguna	10 pengguna
1	1.6	11.8	16.2
2	2.3	10.2	18.4
3	1.6	10.5	21.3
4	2	9.8	18.5
5	1.6	8.2	16.2
Rata-rata	1.683	9.25	16.767

LAMPIRAN 4 – Tampilan Website

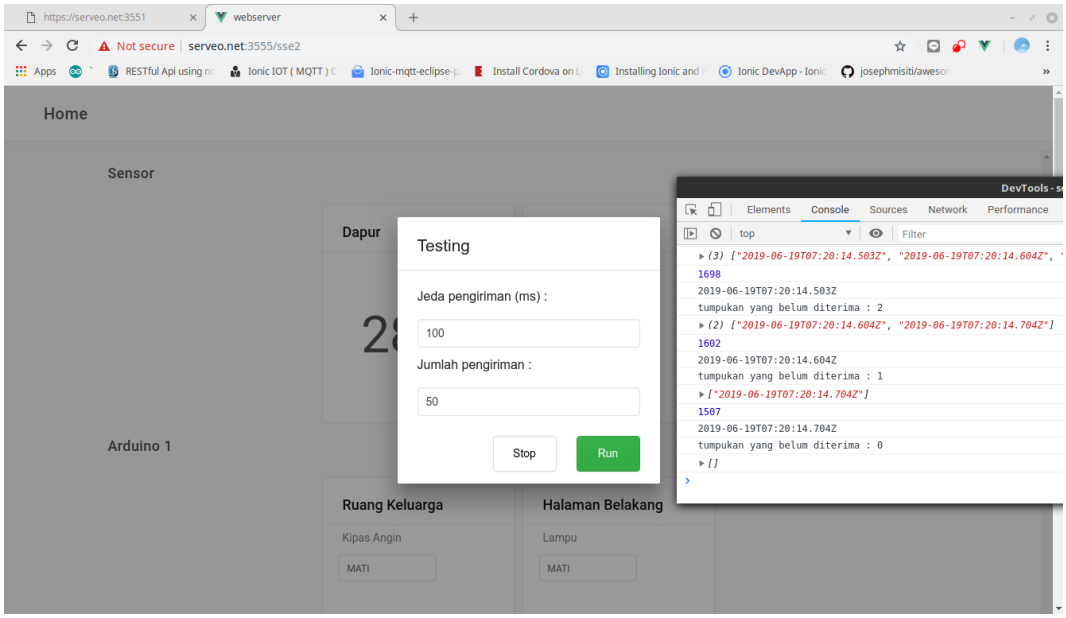


LAMPIRAN 5 – Tampilan Website Saat Pengujian *Response Time*

Saat pengujian HTTP/1.1 SSE dengan jeda pengiriman 100ms melalui internet



Saat pengujian HTTP/2 SSE dengan jeda pengiriman 100ms melalui internet



DAFTAR PUSTAKA

Cirani, Simone, Gianluigi Ferrari, Marco Picone, and Luca Veltri. 2018. *Internet of Things : Architectures, Protocols and Standards*. New Jersey: John Wiley & Sons, Inc.

Hasibuan, Zainal A. 2007. *Metodologi Penelitian Pada Bidang Ilmu Komputer Dan Teknologi Informasi, Konsep, Metode Teknik, Dan Aplikasi*. 2011.

Hillar, Gastón C. 2017. *MQTT Essentials : A Lightweight IoT Protocol : The Preferred IoT Publish-Subscribe Lightweight Messaging Protocol*. Birmingham: Packt.

Abyl. 2018. "Long Polling - Concepts and Considerations." 2018. https://www.ably.io/concepts/long-polling.

Briere, Danny, and Pat Hurley. 2011. *Smart Homes For Dummies*. New Jersey: John Wiley & Sons.

Cirani, Simone, Gianluigi Ferrari, Marco Picone, and Luca Veltri. 2018. *Internet of Things : Architectures, Protocols and Standards*. New Jersey: John Wiley & Sons, Inc.

Elman, Julia, and Mark Lavin. 2014. *Lightweight Django: Using REST, WebSockets, and Backbone - Julia Elman, Mark Lavin*. California: O'Reilly Media, Inc.

Estep, Eliot. 2013. "Mobile HTML5: Efficiency and Performance of WebSockets and Server-Sent Events." Aalto University.

Hillar, Gastón C. 2017. *MQTT Essentials : A Lightweight IoT Protocol : The Preferred IoT Publish-Subscribe Lightweight Messaging Protocol*. Birmingham: Packt.

Ibrahim, Dogan. 2014. "A New Approach for Teaching Microcontroller Courses to Undergraduate Students." *Procedia - Social and Behavioral Sciences* 131 (May): 411–14.

Kayal, Paridhika, and Harry Perros. 2017. "A Comparison of IoT Application Layer Protocols through a Smart Parking Implementation." *Proceedings of the 2017 20th Conference on Innovations in Clouds, Internet and Networks, ICIN 2017*, no. January: 331–36.

Kwan, Joel, Yassine Gangat, Denis Payet, and Remy Courdier. 2016. “An Agentified Use of the Internet of Things.” In *2016 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 311–16. IEEE.

Leach, Paul J., Tim Berners-Lee, Jeffrey C. Mogul, Larry Masinter, Roy T. Fielding, and James Gettys. 1999. “Hypertext Transfer Protocol -- HTTP/1.1.”

Ludin, Stephen, and Javier Garza. 2017. *Learning HTTP/2: A Practical Guide for Beginners - Stephen Ludin, Javier Garza*. Sebastopol: O’Reilly Media, Inc.

MDN. 2019. “Protocol Upgrade Mechanism - HTTP | MDN.” 2019. https://developer.mozilla.org/en- US/docs/Web/HTTP/Protocol_upgrade _mechanism.

Muhammad, Panser Brigade, Widhi Yahya, and Achmad Basuki. 2018. “Analisis Perbandingan Kinerja Protokol Websocket Dengan Protokol SSE Pada Teknologi Push Notification.” *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer (JPTIIK) Universitas Brawijaya* 2 (6): 2235–42.

Örnmyr, Oliver, and Rasmus Appelqvist. 2017. “Performance Comparison of XHR Polling , Long Polling , Server Sent Events and Websockets.” Blekinge Institute of Technology.

Peon, R., and H. Ruellan. 2015. “HPACK: Header Compression for HTTP/2.” https://doi.org/10.17487/RFC7541.

Ramli, Kalamullah, Asril Jarin, and Suryadi Suryadi. 2018. “A Real-Time Application Framework for Web-Based Speech Recognition Using HTTP/2 and SSE.” *Indonesian Journal of Electrical Engineering and Computer Science* 12 (3): 1230.

Rhee, Kyung-Hyune, and Jeong Hyun Yi. 2015. *Information Security Applications: 15th International Workshop, WISA 2014*. Berlin: Springer.

Rochman, Hudan Abdur, Rakhmadhany Primananda, and Heru Nurwasito. 2017. “Sistem Kendali Berbasis Mikrokontroler Menggunakan Protokol MQTT Pada Smarthome.” *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer* 1 (6): 445–55.

Saito, Nobuo, and David Menga. 2015. *Ecological Design of Smart Home Networks : Technologies, Social Impact and Sustainability*. Cambridge: Woodhead Publishing.

Souders, Steve. 2009. *Even Faster Web Sites: Performance Best Practices for Web Developers*. California: O'Reilly Media.

Udayashankara, V, and M S Mallikarjunaswamy. 2009. *Microcontroller*. New Delhi: Tata McGraw-Hill Education.

Vasseur, Jean-Philippe, Adam Dunkels, Jean-Philippe Vasseur, and Adam Dunkels. 2010. "What Are Smart Objects?" *Interconnecting Smart Objects with IP*, January, 3–20.

Wang, Vanessa., Frank. Salim, and Peter. Moskovits. 2013. *The Definitive Guide to HTML5 WebSocket*. Apress.