Indonesian Journal of Electrical Engineering and Computer Science
Vol. 12, No. 3, December 2018, pp. 1230~1238
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v12.i3.pp1230-1238

A Real-time Application Framework for Speech Recognition Using HTTP/2 and SSE

Kalamullah Ramli[1], Asril Jarin[2], Suryadi[3]
[1,2]Department of Electrical Engineering, Faculty of Engineering, Universitas Indonesia, Indonesia
[3]Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia
Depok 16424, Indonesia
[2]Center for Information and Communication Technology, BPPT Puspiptek Serpong, Tangerang Selatan 15314, Indonesia

Article Info

Article history:

Received May 22, 2018
Revised Sep 2, 2018
Accepted Oct 8, 2018

Keywords:

HTTP/2
Network speech recognition
Real-time speech recognition
Server-sent event
WebSocket

ABSTRACT

The performance of network-based speech recognition application is mainly determined by the availability of all speech data received on the server and also the realtimeness in delivering the recognition results from the server. On the basis of full-duplex speech recognition application this paper proposes a real-time application framework for web speech recognition using HTTP/2 protocol and Sender-Sent Events (SSE). A number of experiments were performed to compare the latency of both the application using HTTP/2 plus SSE and the full-duplex application using WebSocket. The results showed that the proposed framework offers better alternative for a web-based speech recognition than the framework using WebSocket.

Corresponding Author:

Kalamullah Ramli,
Department of Electrical Engineering,
Faculty of Engineering, Universitas Indonesia, Indonesia.
Email: kalamullah.ramli@ui.ac.id

1. INTRODUCTION

Automatic speech recognition over the Internet faces packet loss and delay [1]. Packet loss in turn lead to errors in decoding the speech signal to the text [2]. This prevents TCP becoming a good solution for speech recognition over the Internet. TCP ensures all speech data received by the server but introduces delay caused by two main mechanisms, namely additive increase multiplicative decrease (AIMD) and Timeout [3]. To overcome the delay of TCP flow, many researches attempt to seek solutions on the real-time basis, such as multimedia streaming and voice over IP, by modeling TCP in certain circumstances where it may deliver a satisfactory performance [4-6]. Meanwhile, applications, such as one developed by Google emerge to minimize the TCP delay by compressing the source data and optimizing the communication protocol between the client and the server to make high-performance websites [7].

TCP-based speech recognition with speech segmenter on the client side requires acceptable delay to meet users' satisfaction. Our previous work developed a model to investigate this acceptable delay for bahasa Indonesia and to find out a feasible working region of TCP flow on the basis of loss rate and the average round-trip time [8]. In other work [9], the investigation of the acceptable delay was attempted through calculations of the packet delay distribution model [10]. The results indicated that model [8] was more appropriate to use.

In recent years, the solution for a real-time application over TCP is dealt with sending immediately the updates from the server to the client using asynchronous communication techniques, such as polling, long-polling, and streaming [11]. Polling is a technology in which a browser sends HTTP requests in a regular time interval in order to get immediately the updates from the server. A suitable example of applying this technology is to measure the water level and the temperature remotely [12]. A good variant of polling technology is long

polling. This variant emulates a push mechanism from the server. Server holds the request open until an update appears on the server or a timeout [13]. The next technology is streaming. It is a technology that sends a complete request from a browser to the server and let it open indefinitely. Neither the client nor the server needs to close the connection [13].

A more advanced technology is a full-duplex communication using WebSocket. Google developed WebSocket to enable the communication in a bi-directional way between the client and the server on a single TCP-connection [14]. WebSocket is exploited by Alumae in order to develop a full-duplex speech-to-text for Estonian [15]. In this system, the client sends the speech data in some containers and encodings supported by GStreamer framework (e.g., Ogg, MP4, and Speex) to the server. Meanwhile, the server will send the intermediary results, which are called hypotheses, to the client. The system uses a Kaldi online decoder for segmenting the speech data into speech sentences. The speech sentence is then recognized progressively by a speech recognizer to generate hypotheses and a final hypothesis [16].

A significant problem of WebSocket is when it has to pass over a proxy server [17]. A proxy server usually does not allow an idle connection, which is opened for a long time. Therefore, this paper proposes an alternative, a real-time application framework using Hypertext Transfer Protocol version 2.0 (HTTP/2) plus Server-Sent Event (SSE). HTTP/2 is a hypertext transfer protocol that overcomes weaknesses of HTTP/1.1, especially to reduce the application latency [18, 19]. In cooperation with SSE [20], the application enables the server to send updates to the client. The framework using HTTP/2 plus SSE is developed on the basis of Alumae's application [15] in which the WebSocket communication between the client and the server is replaced by combination of HTTP/2 and SSE. Furthermore, experiments are conducted to compare the latency of application developed using current framework, i.e., WebSocket and the proposed framework (HTTP/2 plus SSE).

2. METHODOLOGY

2.1. Hypertext Transfer Protocol version 2.0 (HTTP/2)

HTTP/2 is the new version of Hypertext Transfer Protocol, which is published in May 2015 by Internet Engineering Task Force (IETF) as RFC 7540 [18]. The main goal of HTTP/2 is to make the application faster, simpler, and more robust by providing these following features, i.e., full request and response multiplexing; HTTP header fields compression, request prioritization, and server push. HTTP/2 does not modify semantically the HTTP application because it still uses all main concepts of an HTTP protocol, e.g., HTTP methods, header fields, status codes, and URIs. The most important enhancement of HTTP/2 is the new binary framing layer within an application layer as shown in Figure 1. It makes HTTP messages encapsulated into frames and transferred over TCP channel.

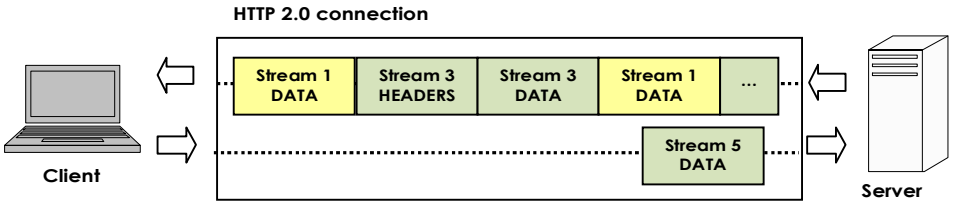

Figure 1. HTTP/2 binary framing layer [22]

All HTTP/2 communications run on a single TCP connection, which is able to bring one or more bi-directional streams. Each stream has a unique pair of identifier and priority number. This pair is used to tag bidirectional message of HTTP to identify stream a frame belongs to. An HTTP message could be as HTTP request or HTTP response. The protocol arranges the frames of different streams and interleaves them when sending the message. At the other end the protocol reassembles them by using the stream identifier which is carried by each frame in its header.

With the framing model, HTTP/2 multiplexes the HTTP request and HTTP response by splitting HTTP messages into frames, interleaving, and reassembling them on the receiver as shown in Figure 2. HTTP/2 only needs one connection per to reduce the latency an[d improve the throughput. However, there is a negative consequence when TCP suffers head-of-blocking or decreasing the congestion window. Fortunately, this drawback can be partially compensated by the advantages of HTTP/2 mechanisms such as header compression and stream prioritization [21].

The possible negative effects of this technique as described in RFC 6202 (2011) [13].

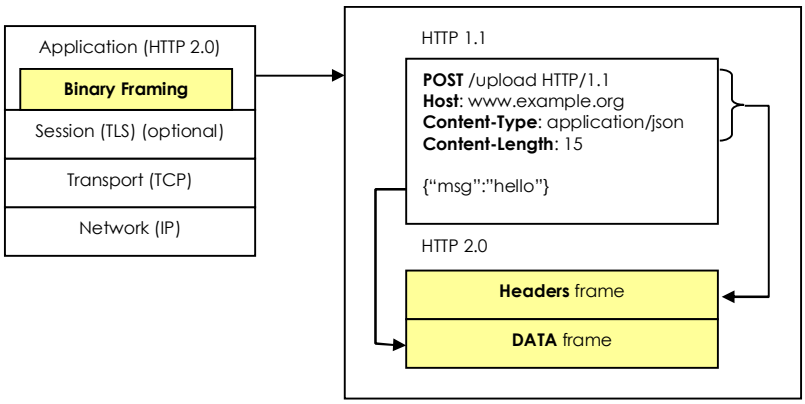

Figure 2. HTTP/2 request- and response multiplexing within a shared connection [22]

2.2. Server-Sent Events (SSE)

Server-Sent Events (SSE) is a technology whereby a browser receives automatic updates from a server via HTTP connection. SSE has EventSource API, which is standardized as part of HTML5 by World Wide Web Consortium (W3C) [20]. The specification of SSE defines an API that allows servers to push data to Web pages over HTTP in the form of Document Object Model (DOM) events. The data is encoded as "text/event-stream" content and pushed by using an HTTP streaming mechanism. However, the specification suggests to disabling the HTTP chunking for serving event streams unless the rate of messages is high enough to avoid.

2.3. Application Framework with HTTP/2 plus SSE

The proposed real-time application framework using HTTP/2 plus SSE is developed as a modification of a full-duplex application framework using WebSocket, as shown in Figure 3. The framework consists of two main components: client and server. A client acts to provide speech data and presents the recognition results to the user whereas a server acts to decode and to hypothesize the received speech data as well as to normalize them. The server is divided into a master-server and one or more workers. Master-server works in a Tornado Framework [23] and manages a set of workers connected with their status; forwarding the received speech data to the worker and pushing immediately the recognition results to the client. The worker serves the speech recognition using GStreamer Online Decoder Plugin from Kaldi Toolkit [24].

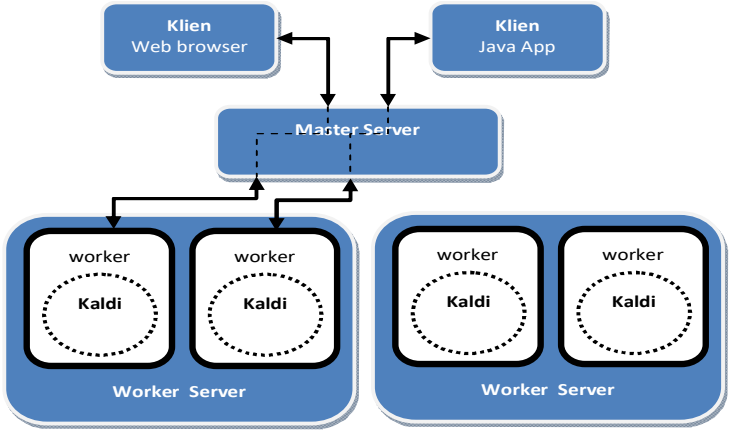

Figure 3. Full-duplex application framework from Alumae

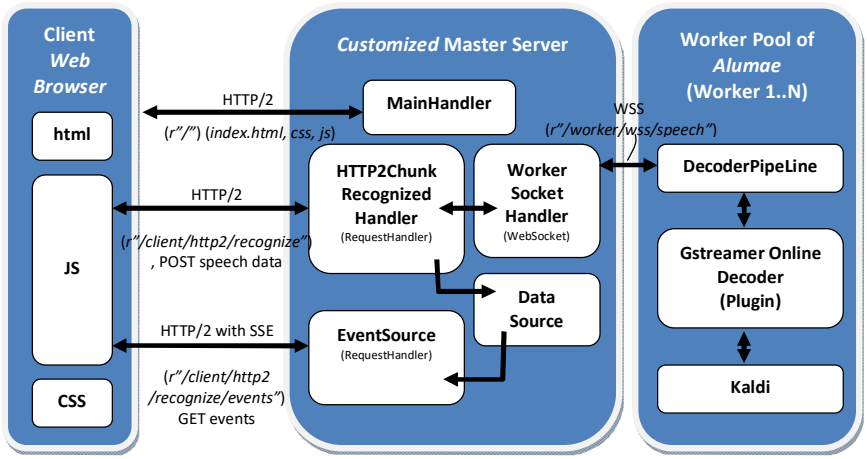

Figure 4. Real-time application framework for speech recognition using HTTP/2 plus SSE

Figure 4 illustrates the real-time application framework using HTTP/2 plus SSE. The communication between client and master-server is done by HTTP/2 protocol in cooperation with Server-Sent Events (SSE) on a single TCP connection as shown in Figure 4. HTTP/2 multiplexes two streams coming from two HTTP/2 requests. The first request is sent to ask for the speech recognition service from the server and for sending the speech data to be recognized. The second request is sent for SSE service in order to push the intermediary recognition results from the server which is called hypotheses. The user may send the speech data in any container and decoding in any formats supported by GStreamer, i.e., Ogg, MP4, Speex, etc. The recognition results are sent to the client in JSON format.

For interaction with the client, master-server provides three handlers, i.e., 'MainHandler', 'HTTP2ChunkRecognizedHandler' and 'EventSource'. MainHandler handles the first request of the client in order to get the web page and other resources, i.e., stylesheet files dan java-script files. HTTP2ChunkRecognizedHandler manages the HTTP request for recognition service, which is executed by a worker as back-end process on the server. The speech data is forwarded by the master-server to the worker. To perform its work, the worker has some modules namely Decoder PipeLine, GStreamer Online Decoder (plugin) dan Kaldi (ASR). For multiple recognition services, the workers are placed in a pool. The master-server takes one of the available workers when client requests connection.. Similarly in Alumae's application, each worker is connected with the master-server in a full-duplex communication so that they can run either in a local host or in a remote host. This communication is handled by the master-server handler called 'WorkerSocketHandler'. Furthermore, the master-server uses the handler of EventSource for SSE service. Since SSE request is sent by the client, handler of EventSource will get the events to be published through monitoring the updates in 'DataSource'. DataSource stores the hypotheses from worker.

2.4. Experiment

There are three steps in implementing the proposed framework into the Alumae's application: providing HTTP/2 server support in master-server; providing three handlers in master-server (MainHandler, HTTP2ChunkRecognizedHandler, and EventSource); and providing the client programs for HTTP/2 plus SSE supported Browsers.

Figure 5 shows an example of Web page that we use for the experiments. In this Web page, there are features we need to conduct performance comparison between full-duplex application and HTTP/2 plus SSE application. Those are audio playback; buttons for sending, the recognition service either using HTTP/2 plus SSE or WebSocket; part of page for speech data information, connection status and some measurements for latency, connection speed and real-time-factor (RTF); and part of page for presenting hypotheses and final result of the recognition.

To evaluate whether the proposed framework is more efficient than the full-duplex framework we developed an experimental environment by using an *ns*-3 based emulation platform [25], as shown in Figure 6. The environment employed a network simulation that is also used for validating the analytical model and explained in our previous work [8]. To simulate a real network a tap device and an ns-3 tap bridge is employed in that environment. The real node of client runs on the virtual-host (Ubuntu 16.04 LTS) in a virtual system connected by a linux-bridge to the tap device prepared on the linux-host. The tap device of the client is connected to the simulated network in a TapBridge UseBridge mode. Meanwhile, the real node of server runs

on the linux-host by using a tap device connected to the simulated network in a TapBridge UseLocal mode. The detailed information about the ns-3 tap-bridge modes can be found in *ns*-3 documentation [26].

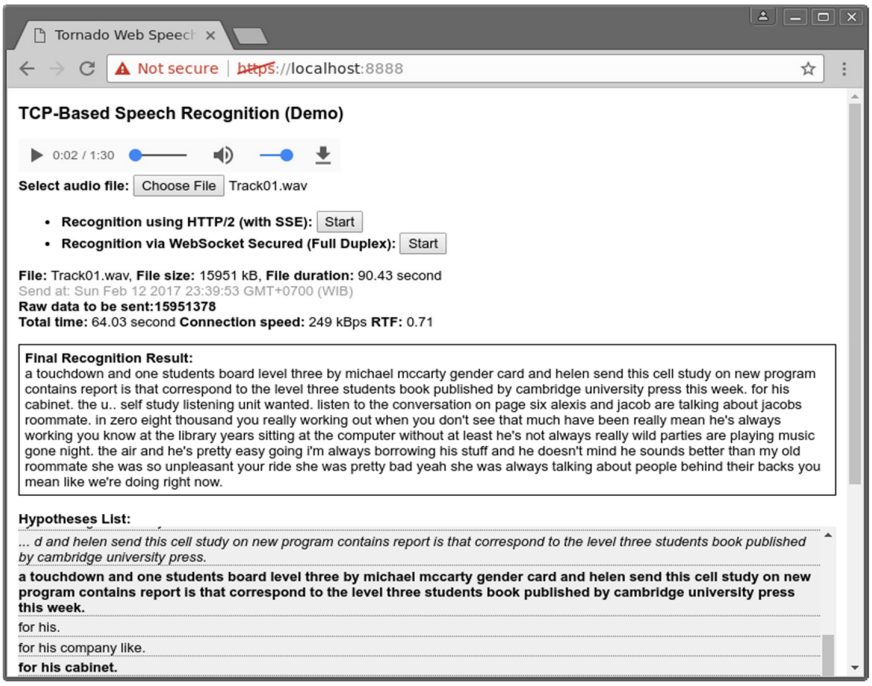

Figure 5. Application website of TCP-based speech recognition

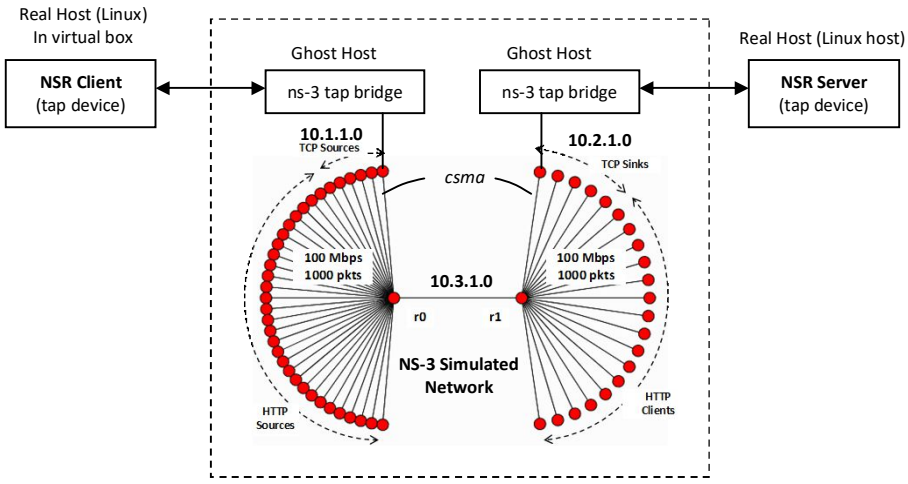

Figure 6. Experimental environment on simulated ns-3 network

Furthermore, the experiments are performed in four simulated network settings to obtain various propagation delays and loss rates, as shown in Table 1. For each setting, we used ten speech data samples of English conversations (WAV files). These samples are not linear so that they do not have relation to the linearity of the application latencies occurred. Nevertheless, this experiment focused on the comparison between two frameworks (HTTP/2 plus SSE and WebSocket).

Table 1. Four parameter settingsof simulated ns-3 network

Set	# of Sources		Bottleneck Link			Parameter App	
	TCP	HTTP	P.Delay (ms)	B.w. (Mbps)	Buffer (pkts)	Delay (ms)	Loss Rate
1	9	40	40	3.7	50	120	0,4% - 0,5%
2	5	30	40	3.7	50	120	0,2% - 0,3%
3	9	40	5	5	100	50	0,4% - 0,5%
4	5	30	5	5	100	50	0,2% - 0,3%

3. RESULTS AND DISCUSSION

There are two things that make the proposed framework is different from the full-duplex framework: It uses HTTP/2 cooperating with SSE; Its client is implemented in the browser as HTML and javascript. All communication are handled using protocol HTTP/2 and the speech data is sent by browser using XMLHttpRequest and POST method. Since HTTP/s is a new protocol and the modern browsers such as Chrome, Firefox may not allow it to run without security the application of this framework is usually run on the secured communication (TSL/SSL, HTTPS).

Meanwhile, the full-duplex framework using WebSocket has following characters: The client is realized in python program and running in a terminal; WebSocket is not dependent on HTTP. It only uses HTTP for handshaking to get the same connection of TCP; WebSocket communication can also be secured by using TSL/SSL over WSS.

The results of each experiment settings are depicted by Figure 7. Overall, the measurement results show that the application latencies of HTTP/2 plus SSE and WebSocket are comparable.

The differences are occurred due to different settings in the propagation delay, the loss rate, and the average round-trip time, as shown in Figure 8. The Loss rate and the round-trip time are the main parameters of TCP-delay function.

Based on this comparable result three points could be revealed as to why the latency of HTTP/2 plus SSE is not more efficient than WebSocket even though HTTP/2 has some advantages to reduce the latencies. Those are:

a. The application only multiplexed two streams triggered by two HTTP/2 requests, i.e., request to the speech recognition handler and request to SSE handler, as shown in Figure 9. The more number of streams multiplexed the more efficient the system.
b. The compression of HTTP header fields by HPACK (Header Compression for HTTP/2) is less useful because the experimental application does not use a large meta-data and it does not have cookies. This feature will have the significant effect when the application multiplexes more requests so that it compresses more headers from those request and prevents the repeated header fields.

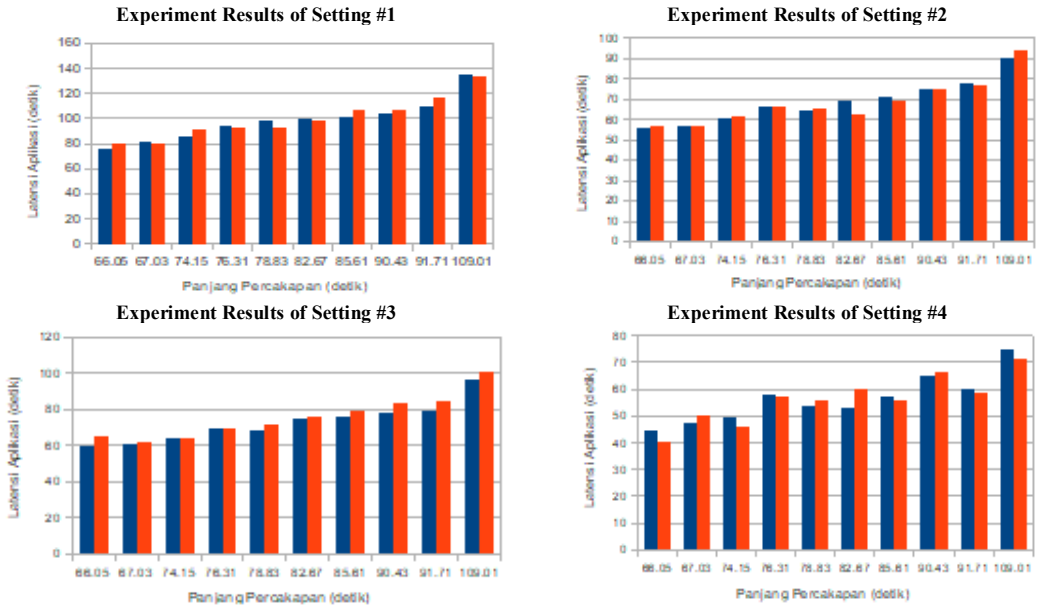

Figure 7. Latency of Application using HTTP/2 plus SSE versus WebSocket

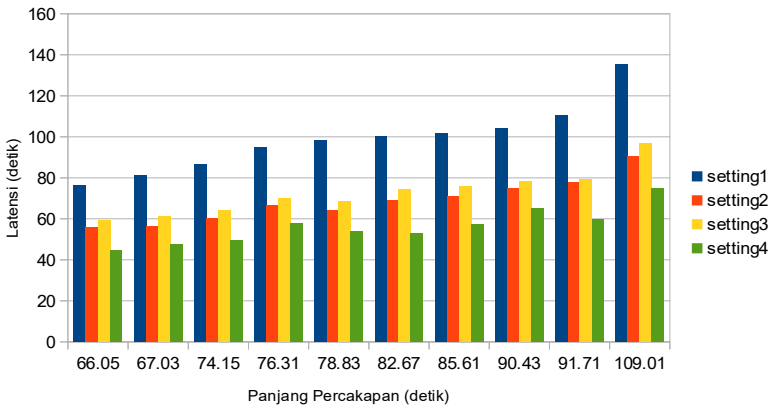

Figure 8. Latencies comparison between HTTP/2 plus SSE vs. WebSocket in each experiment settings

c. The compression of HTTP header fields by HPACK (Header Compression for HTTP/2) is less useful because the experimental application does not use a large meta-data and it does not have cookies. This feature will have the significant effect when the application multiplexes more requests so that it compresses more headers from those request and prevents the repeated header fields.

Last, we examine whether the HTTP/2 plus SSE framework performs better than WebSocket if the application runs over proxy server. A proxy server SQUID 3.5.12 [27] is used.

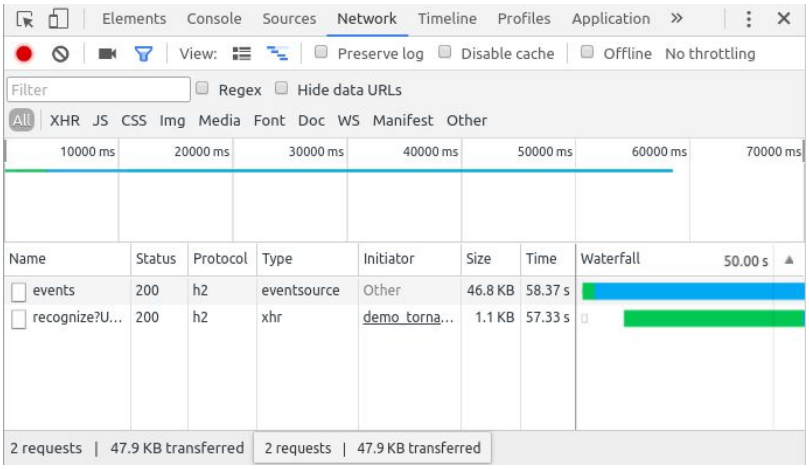

Figure 9. Multiplexing of two streams (HTTP/2 request-response and Server-Sent Events)

Figure 10 shows the WiFi-LAN based environment that consists of a client host with IP address192.168.100.102, a proxy host with IP address 192.168.100.104 and a server. Proxy-server runs on port 3128 while the server proceeds on port 8888. The client is represented by Google Chrome browser, which is set to send the data over the proxy-server with the address 192.168.100.104:3128. All experiment rules are made in the configuration file 'squid.conf'. Then, the experiments are conducted in two scenarios, i.e., running application over the secured connection (HTTPS or WSS) and via the ordinary connection (HTTP or WS).

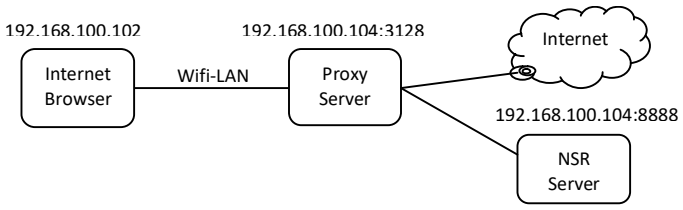

Figure 10. Experiment environment on network with proxy-server

Two conclusions can be drawn from this particular experiment. When both application using HTTP/2 plus SSE and using WebSocket are passed via SSL/TSL connection (HTTPS/WSS) then the proxy server allows their traffic pass through TCP tunnel, and no proxying or caching is needed. Meanwhile when both applications run through other than SSL/TSL connection, then the WebSocket connection is blocked, but the communication using HTTP1.1 plus SSE stays connected. In the latter scenario HTTP/2 is not applicable.

4. CONCLUSION

In this paper, we propose a real-time application framework using HTTP/2 plus Server-Sent Event (SSE). The experiments are conducted to compare the latency of this proposed framework against the latency of a full-duplex application using WebSocket. The results conclude that the latencies from both frameworks are comparable. However, based on the advantages of HTTP/2 protocol and also the reason that the proxy server blocks the WebSocket communication especially in idle situations our proposed framework offers better alternative for a real-time web-based speech recognition than using WebSocket.

ACKNOWLEDGEMENTS

This article's publication is supported by the United States Agency for International Development (USAID) through the Sustainable Higher Education Research Alliance (SHERA) Program for Universitas Indonesia's Scientific Modeling, Application, Research and Training for City-centered Innovation and Technology (SMART CITY) Project, Grant #AID-497-A-1600004, Sub-grant #IIE-00000078-UI-1.

REFERENCES

[1] Van Sciver J, Ma JZ, Vanpoucke F. *Investigation of speech recognition over IP channels*. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). Orlando, Florida, USA. 2002.
[2] Carmona JL. Coded-speech recognition over IP networks. Disertation, Dpto. Teoria de la Senal, Telematica Comunicaciones, University of Granada; 2009.
[3] Stevens WR, Wright GR. TCP/IP illustrated. Massachuset: Addison-Wesley Pub. Co. 1994.
[4] Brosh E., Baset SA, Misra V, Rubenstein D, Schulzrinne H. The Delay-Friendliness of TCP for Real-Time Traffic. *IEEE/ACM Transactions on Networking*. 2010; 18(5): 1478-1491.
[5] Wang B, Kurose J, Shenoy P, Towsley D. Multimedia streaming via TCP: An analytic performance study. *ACM Transactions on Multimedia Computing, Communications and Applications*. 2008; 4(2): 1-22.
[6] Jinyao Y, Muhlbauer W, Plattner B. Analytical Framework for Improving the Quality of Streaming Over TCP. *IEEE Transactions on Multimedia*. 2012; 14(6): 1579-1590.
[7] Souders S. High Performance Web Sites. *Queue*. 2008; 6(6): 30-37.
[8] Jarin A, Fahmi H, Suryadi, Ramli K. Development of Modified Analytical Model for Investigating Acceptable Delay of TCP-Based Speech Recognition. *Advanced Science Letters*. 2017; 23(4): 3654-3659.
[9] Brosh, E., Baset, S. A., Misra, V., Rubenstein, D., Schulzrinne, H. The delay-friendliness of TCP for real-time traffic. *IEEE/ACM Transactions on Networking (TON)*, 2010; 18(5): 1478-1491.
[10] Jarin, A., Suryadi, S., Ramli, K. Packet Delay Distribution Model for Investigating Delay of Network Speech Recognition. *Indonesian Journal of Electrical Engineering and Computer Science*. 2017; 5(1): 11-18.
[11] Lubbers P, Greco F. HTML5 WebSocket: A Quantum Leap in Scalability for the Web. *Websocket Org*. 2010.
[12] Pimentel V, Nickerson BG. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Internet Computing*. 2012; 16(4): 45-53.
[13] Loreto S, Saint-Andre P, Salsano S, Wilkins G. Known issues and best practices for the use of long polling and streaming in bidirectional HTTP. *Internet Engineering Task Force (IETF)*. 2011; RFC 6202.
[14] Fette I, Melnikov A. The websocket protocol. *Internet Engineering Task Force (IETF)*, 2011; RFC 6455.
[15] Alumäe T. *Full-duplex Speech-to-text System for Estonian*. International Conference Human Language Technologies - The Baltic Perspective. Kaunas, Lithuania. 2014; 3-10.

[16] Povey D, Ghoshal G, Boulianne G, Burget L, Glembek O. *The Kaldi Speech Recognition Toolkit*. 2011; Available: http://kaldi.sourceforge.net.
[17] Lubbers P. How HTML5 Web Sockets Interact With Proxy Servers. *InfoQ*. 2010; Available: https://www.infoq.com/articles/Web-Sockets-Proxy-Servers.
[18] Belshe M, Peon R, Thomson M. Hypertext Transfer Protocol version 2 (HTTP/2). *Internet Engineering Task Force (IETF)*. 2015; RFC 7540.
[19] Stenberg D. HTTP2 explained. *ACM SIGCOMM Computer Communication Review*. 2014; 44(3): 120-128.
[20] Hickson I. Server-Sent Events. *W3C*. 2011; Available: https://www.w3.org/TR/2011/WD-eventsource-20110208/
[21] Grigorik I. High Performance Browser Networking: What every web developer should know about networking and web performance. *O'Reilly Media*. 2013.
[22] Grigorik I. Making the web faster with HTTP 2.0. *ACMQUEUE*. 2013; 56(12): 42-49.
[23] Tornado. Tornado Web Framework. 2016; Available: http://www.tornadoweb.org/en/stable/
[24] Ghoshal A, Povey D. The Kaldi Speech Recognition Toolkit. *IEEE Signal Processing Society*. 2012.
[25] Ramli, K., Jarin, A. New ns-3-based Emulation Platform for Performance Evaluation of TCP-based Speech Recognition. *Indonesian Journal of Technology*. 2018; 9(4): 852-861.
[26] NS-3 Consortium. Tap Bridge Model [Devices]. *NS-3 Documentation*. 2011; Available: https://www.nsnam.org/docs/ release/3.9/doxygen/group___tap_bridge_model.html
[27] SQUID. Squid 3.5.12. *squid-cache org*. 2015; Available: http://www.squid-cache.org/