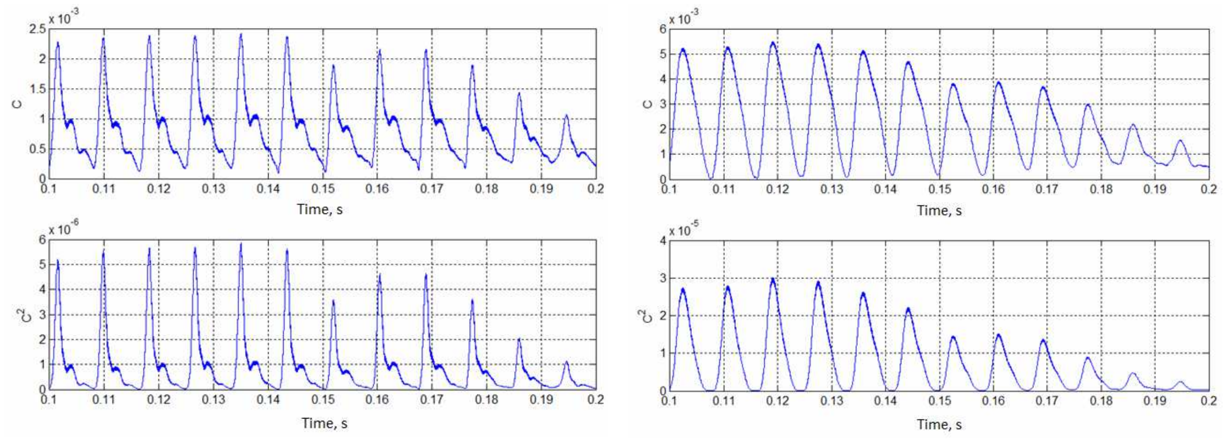

a) parameters wavelet: B =1; f_0 =1.5 b) parameters wavelet: B =2; f_0 =0.5

Fig. 5. Two-dimensional slices of the three-dimensional skeylogramm

Thus, the application of wavelet transformation supplements signal characteristics obtained by the usual statistical methods (in particular spectral), as well as assesses the scaling parameter of signals. Our choice of the mother complex Morlet wavelet and the results of experimental research of parameters of vowel sounds, obtained by that wavelet transformations, allow managing the tuning of the wavelet to obtain a fine structure of the signal of a particular speaker in the real time. Additionally, you can use the wavelet transformations for clearing against harmful interference of measurement harmonic and chirp signals.

REFERENCES

1. Соловьев, В.И. Спектральный анализ и современные речевые технологии. / В.И.Соловьев, О.В. Рыбальский, В.К. Железняк // Вестн. Полоц. гос. ун-та. Сер. С. – 2014. –№ 4. – С. 2–6.
2. Добеши, И. Десять лекций по вейвлетам / И. Добеши ; пер. с англ. – Ижевск : НИЦ «Регулярная и хаотическая динамика», 2001. –464 с.
3. Daubechies, Ingrid. Ten lectures on wavelets / Ingrid Daubechies. – SIAM, Philadelphia, 1992. – 357 p.
4. Малла, С. Вейвлеты в обработке сигналов. / С. Малла ; пер. с англ. – М. : Мир, 2005. – 671 с.
5. Дьяконов, В.П. Вейвлеты. От теории к практике / В.П. Дьяконов. – М. : Солон-Пресс, 2004.– 400 с.
6. Витязев, В.В. Вейвлет-анализ временных рядов : учеб. Пособие / В.В. Витязев. – СПб. : Изд-во С.-Петербург. ун-та, 2001. – 58 с.
7. Addison, Paul S. Illustrated wavelet transform handbook. Introductory Theory and Applications in Science, Engineering, Medicine and Finance / Paul S. Addison. – Bristol : Institute of Physics Publishing, 2002. – 400 p.

UDC 004.9+004.773

FULL-DUPLEX NETWORK COMMUNICATION USING THE WEBSOCKET PROTOCOL

ARTUR DRYOMOV, ARKADY OSKIN
Polotsk State University, Belarus

The paper discusses current methods of full-duplex network communication and a powerful alternative called the WebSocket protocol. The protocol itself is described and compared to current solutions.

The modern world wide web as we know it at this point was established a while ago, including but not limited to technologies that are used as the basis of the web. The paradigm used for the Hypertext Transfer Protocol (HTTP) can be called as *Request-Response*. When a user goes to a web page he sends a *Request* to a web server. The web server sends a *Response* as a result of this interaction. Nothing is really happening until the user decides to navigate to the next destination page. The current web requires a lot more than this. Users want to load their content instantly and many web applications are based on the single page architecture, where content is updated on the single page, without the page reloading.

There are several techniques that allow this kind of interaction with a web content at this point. The most common one is so called *Long Polling*. When using this technique a client, which can be any HTTP client such as a web browser or any HTTP-based application, opens a HTTP connection to the web server using a request which keeps it open until sending a response. This technique works quite well, but carries a major issue. This way a client should carry HTTP overhead, even if it is not necessary. It can be a blocker for low latency applications, especially mobile devices which can have access only to an unstable wireless cellular network connection such as GPRS (General Packet Radio Service), HDSPA (High-Speed Downlink Packet Access) or LTE (Long-Term Evolution) [1].

The WebSocket protocol defines an API (Application Programming Interface) which allows direct full-duplex data communication. The great advantage of the protocol is that it is not HTTP-based, but utilises TPC (Transmission Control Protocol) under the hood. TCP is a low-level protocol which provides WebSocket with a benefit of high performance without the HTTP overhead. The WebSocket connection is full-duplex. Basically it means that both sides of the connection can communicate at the same time. The great and popular example of a full-duplex connection is a telephone. Both sides using the telephone can use the connection simultaneously. Another factor providing WebSocket its efficiency is using a single persistent TCP connection for communication purposes. There is no need to reload the web page in a web browser, move to another page or apply a long polling technique. Because the TCP connection is used the impact on network usage is very low comparing to methods enumerated above. The last but not the least benefit is port usage. WebSocket communications are evaluated using the port number 80 which is used for web connections. This is a huge benefit for environments blocking all non-web related network connections using such tools as firewalls [2].

The protocol itself was widely adopted at the moment of writing this paper. All major web browsers such as Google Chrome, Microsoft Internet Explorer, Mozilla Firefox and Apple Safari already support WebSocket communication. In other words it is possible to use WebSocket protocol on the web application level, without using any complicated libraries or frameworks.

A WebSocket connection can be identified using URI (Uniform Resource Identifier) with a specific to the protocol schemes such as WS (WebSocket) and WSS (WebSocket Secure). This naming policy follows HTTP policy which itself can be defined as protocol schemes HTTP and HTTPS (HTTP Secure or HTTP over SSL, i. e. encrypted HTTP). The only semantics for such scheme is to open a WebSocket connection [3].

The protocol itself consists of two different but connected parts. The first one is a handshake which opens or closes a WebSocket connection. The second one is a data transfer which provides the data exchange itself.

The handshake responsible for opening a WebSocket connection is very important. The protocol was designed in such a way that it is HTTP-compatible. This is made for unified network port usage. In other words both WebSocket and HTTP clients can interact with a single server using a single port.

There is an example of an opening handshake. A WebSocket client sends to a server such request. The request header fields can be specified in any order, but the content itself is important.

```
GET /chat HTTP/1.1
Host: server.domain.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: aBcDeFgHiJkLmNoPqRsTuVwXyZ==
Sec-WebSocket-Protocol: chat, chat-extra
Sec-WebSocket-Version: 13
```

The client includes the *Host* header field where it specifies a used domain. This way both the client and the server have an ability to verify that they are using the same host for communication. The *Connection* header field specifies the HTTP Upgrade method. So the server has knowledge about the necessary protocol switching. The protocol for switching is specified at the *Upgrade* header field. Other header fields specify WebSocket-related options and characteristics [2].

– *Sec-WebSocket-Protocol* — this header field contains a list of protocol extensions used in the communication process. In practice it is used to specify which subprotocols used for communication are supported and are acceptable for a particular WebSocket client. After the server evaluates the initial request it will respond with a selected communication subprotocol.

– *Sec-WebSocket-Key* — this header field is used for validation purposes. When a client sends an initial handshake to a server it sends a partial validation key in this field. After the server receives the client request, it must use it to create its own key and send it back to the client.

– *Sec-WebSocket-Extension* — this header field can be used to specify WebSocket protocol-level extensions required by a client to communicate with a server.

– *Sec-WebSocket-Accept* — this header field typically is used by a server to confirm that the WebSocket connection is created successfully.

– *Sec-WebSocket-Version* — this header field is required to specify which WebSocket protocol version is intended to use for communication purposes. This can be used to provide some sort of backwards compatibility if a server supports multiple WebSocket protocol versions. Or, if a server does not support specified by a client protocol version, the client connection request can be rejected. In this case a server should send a supported protocol version in a response.

In exchange for the client request the WebSocket server must send a response. There is an example of such a response below.

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: ZyXwVuTsRqPoNmLkJiHgFeDcBa=
Sec-WebSocket-Protocol: chat

As we can see the server-produced response is quite simple. First, the server indicates that it had switched the protocol utilised from HTTP to WebSocket as indicated in the *Upgrade* header field. The *Sec-WebSocket-Accept* contains a server-generated security key which can be used by the client to ensure connection integrity and indicates itself that the server successfully accepted the connection. By the *Sec-WebSocket-Protocol* the server indicates the chosen protocol supported by the server and used by the following communication.

When both the server and the client have exchanged their requests and response it is possible to begin the data transfer procedure. The communication is full-duplex, i. e. two-way. The communication channel has an ability to provide simultaneous reading and writing at the same time.

The data transmitted between communication sides in terms of the protocol are called messages. From a protocol perspective a message can consist of one or more the so called frames. This allows operating messages with unknown resulting message length. A frame itself has a special type [2].

– Textual frame — interpreted as a Unicode text with the UTF-8 encoding.
– Binary frame — raw data which is associated by the communication sides.
– Control frame — protocol-related private information not intended for communication sides direct usage. Example of such data is a control sequence indicating that the connection should be closed.

A frame itself has a special predefined by a protocol structure. It is somewhat similar to a TCP packet structure, but has different fields such as:

– Final frame indicator.
– Opcode.
– Mask.
– Mask key.
– Payload length.
– Payload data.
– Extension data.
– Application data.

The handshake responsible for closing a WebSocket connection is as important as the opening handshake, but it is much simpler.

Either a WebSocket server or a WebSocket client can send a control frame with a closing frame which consists only of a special opcode, i. e. control sequence. After sending the frame the receiver of this message sends all the remaining data and sends a closing response. Only after receiving all remaining data and the closing confirmation the closing initiator closes the WebSocket connection. This way no data is lost and both sides of the connection are agreed in their intentions. Basically the closing handshake is quite similar to TCP closing handshake.

The WebSocket protocol provides a powerful, but yet simple and efficient way of full-duplex network communication. It solves a variety of issues specific to current full-duplex solutions [4]:

– The web server is forced to use multiple different underlying TCP connections for each of its clients. There are at least two connections involved. The first one is used for incoming messages and the second one is used for outgoing ones.
– The web client is responsible for tracking and associating incoming and outgoing connections to track messages properly.
– The HTTP protocol has a high overhead for a full-duplex communication.

The WebSocket protocol not only solves these issues, but provides a great base for future improvements and high-level interaction protocols.

REFERENCES

1. Almasi, A. Evaluation of WebSocket Communication in Enterprise Architecture / A. Almasi, Y. Kuma ; University of Gothenburg. – Gotenborg, 2013. – 12 p.
2. Fette, I. RFC 6455 – The WebSocket Protocol [Electronic Resource] / I. Fette, A. Melnikov. – Mode of access: http://tools.ietf.org/pdf/rfc6455.pdf. – Date of access: 15.01.2016.
3. Berners-Lee, T. RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0 [Electronic Resource] / T. Berners-Lee, R.T. Fielding, H. Frystyk Nielsen. – Mode of access: http: tools.ietf.org/pdf/rfc1945.pdf. – Date of access: 16.01.2015.
4. Hadden, R. A. A WebSocket-based Approach to Transporting Web Application Data / R.A. Hadden ; University of Cincinnati. – Cincinnati, 2014. – 37 p.