Technologie Internetu

Protokół Websocket

Aleksander Denisiuk

denisjuk@pja.edu.pl

Polsko-Japońska Akademia Technik Komputerowych

Wydział Informatyki w Gdańsku

ul. Brzegi 55

80-045 Gdańsk

Protokół Websocket

Najnowsza wersja tego dokumentu dostępna jest pod adresem `http://users.pja.edu.pl/~denisjuk/`

Przykład połączenia

```
var socket
  = new WebSocket("ws://echo.websocket.org/");
```

Cztery zdarzenia: `open`, `close`

```
socket.onopen = function() {
  alert("Nawiązazno połączenie.");
};

socket.onclose = function(event) {
  if (event.wasClean) {
    alert('Połączenie poprawnie zamknięte');
  } else {
    alert('Połączenie zostało zerwane');
  }
  alert('Kod: ' + event.code + ' powód: '
                             + event.reason);
};
```

Cztery zdarzenia: `message`, `error`

```
socket.onmessage = function(event) {
  alert("Otrzymano dane: " + event.data);
};

socket.onerror = function(error) {
  alert("Błąd " + error.message);
};
```

Wysyłanie danych

- metoda `send()`
- wysyłać można dowolne dane
- przykłady:

```
socket.send("Hello");
socket.send(form.elements[0].file);
```

Zamykanie połączenia

- metoda `close()`
- przykład:

```
socket.close();
```

Wsparcie przez przeglądarki

- Firefox 6
- Google Chrome 14
- Opera 12.10
- Internet Explorer 10

Realizacje po stronie serwera

- Rozszerzenia dla rozmaitych języków programowania
- W tym:
  - C/C++
  - Java
  - Perl
  - PHP
  - Node.js
  - Python
  - Ruby
  - Erlang

Emulacja protokołu

- SockJS
  - Node.js
  - Ruby
  - Django
  - etc
- Socket.IO

Nawiązanie połączenia. Protokół HTTP

```
GET ws://echo.websocket.org/ HTTP/1.1
Origin: http://websocket.org
Cookie: __utma=99as
Connection: Upgrade
Host: echo.websocket.org
Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
Upgrade: websocket
Sec-WebSocket-Version: 13
```

Nawiązanie połączenia. Nagłówki

- `Connection`, `Upgrade` — propozycja przejścia na websocket
- `Origin` — domena i port, skąd pochodzi zapytanie
- `Sec-WebSocket-Key` — losowy klucz, generowany przez przeglądarkę: 16 bajtów, Base64
- `Sec-WebSocket-Version` — wersja protokołu (obecnie jest trzynasta)

Websocket a XMLHttpRequest

- Nie można utworzyć takiego zapytania za pomocą `XMLHttpRequest`
- Obiekt `XMLHttpRequest` nie może ustawić w zapytaniu nagłówków propozycji przejścia na WebSocket

Websocket a funkcja fetch()

- Nie można utworzyć takiego zapytania za pomocą `fetch()`
- Funkcja `fetc()` nie może ustawić w zapytaniu nagłówków propozycji przejścia na WebSocket

Nawiązanie połączenia. Odpowiedź serwera

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Fri, 10 Feb 2012 17:38:18 GMT
Connection: Upgrade
Server: Kaazing Gateway
Upgrade: WebSocket
Sec-WebSocket-Accept: rLHCkw/SKsO9GAH/ZSFhBATDKrU=
```

- Dalsza wymiana danych odbywa się na tym samym łączu TCP/IP, na tym samym porcie, ale z użyciem protokołu WebSocket
- HTTP już się nie używa

Rozszerzenia

- Nagłówek `Sec-WebSocket-Extensions` oznacza użycie dodatkowych możliwości przekazywania danych
- Przykład:
  `Sec-WebSocket-Extensions: deflate-frame`
- Ustawia się automatycznie przez przeglądarkę

Subprotokoły

- Nagłówek `Sec-WebSocket-Protocol` oznacza, że będą przesyłane dane według specyficznego protokołu
- Przykład:

```
var socket =
    new WebSocket("ws://echo.websockets.org",
                  ["soap", "wamp"]);
```

- `Sec-WebSocket-Protocol: soap, wamp`

Przykład zapytania

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://example.com
Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==
Sec-WebSocket-Version: 13
Sec-WebSocket-Extensions: deflate-frame
Sec-WebSocket-Protocol: soap, wamp
```

Przykład odpowiedzi

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: hsBlbuDTkk24srzEOTBUlZAlC2
Sec-WebSocket-Extensions: deflate-frame
Sec-WebSocket-Protocol: soap
```

wss

- wss jest protokołem WebSocket nad HTTPS

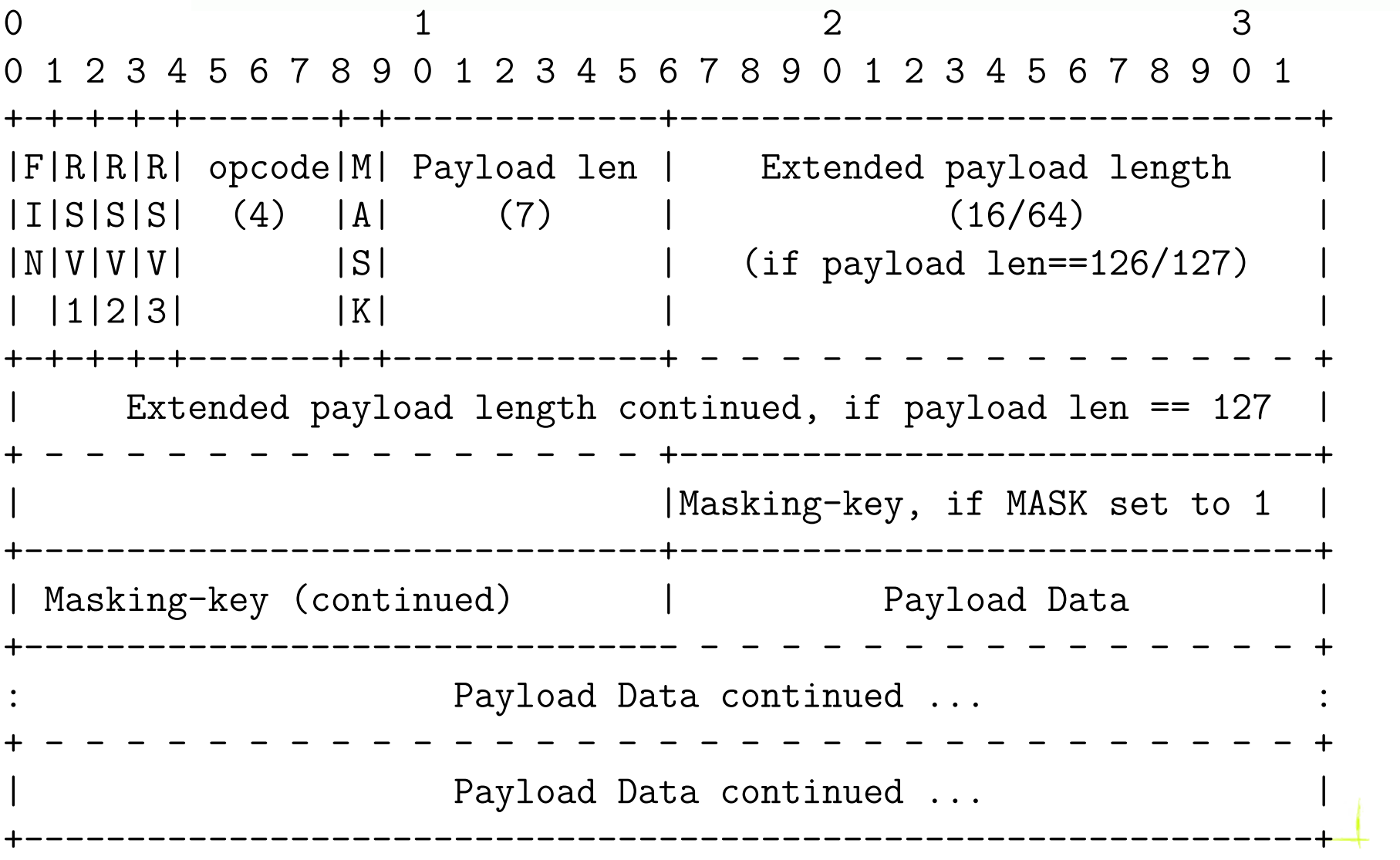
```
var socket =
  new WebSocket("wss://www.websockets.org");
```

- dane są szyfrowane
- większe szanse nawiązać połączenie przez proxy

Ramki WebSocket

- Ramki (Frame) protokołu mogą być
  - kontrolne (sprawdzenie połaczenia, zamknięcie połaczenia)
  - ramki danych

Struktura ramki WebSocket



Ramka. Pierwsze cztery bity

- Bit `FIN`: dla ostatniej ramki 1, dla wszystkich poprzednich 0
- Bity `RSV1`, `RSV2`, `RSV3` — powinny być zerowe, jeżeli nie używa się rozszerzeń protokołu. Rozszerzenia ustawiają te bity po-swojemu

Ramka. Kod operacji (opcode)

- 4 bity, określa typ ramki
  - 0x1 — ramka tekstowa
  - 0x2 — ramka binarna
  - 0x3–0x7 — zarezerwowano dla przyszłych typów danych
  - 0x8 — zamykanie połączenia tą ramką
  - 0x9 — PING
  - 0xA — PONG
  - 0xB–0xF — zarezerwowano dla przyszłych ramek sterujących
  - 0x0 — ramka-kontynuacja (ten sam typ, co w poprzednim fragmencie ramki)

Ramka. Bit maski (MASK)

- jeżeli bit jest ustawiony, to dane są maskowane

Ramka. Rozmiar danych komunikatu (payload length)

- 7 bitów, jeżeli rozmiar jest mniej niż 126
- jeżeli w polu jest zapisane 126, to następne 2 bajty zawierają 16-bitowy rozmiar danych komunikatu
- jeżeli w polu jest zapisane 127, to następne 8 bajtów zawierają 64-bitowy rozmiar danych komunikatu

Ramka. Klucz maski (Masking-key)

- 4 bajty, jeżeli bit maski jest jedynką
- nie ma tego pola, jeżeli bit maski jest zerem

Ramka. Dane (Payload Data)

- długość zgadza się z długością, wskazaną w nagłówku

Ramka. Przykłady

- `0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f` (Hello)
- „Hello World” w trzech częściach

```
0x01 0x03 0x48 0x65 0x6c 0x6c 0x6f
0x00 0x01 0x20
0x80 0x05 0x57 0x6f 0x72 0x6c 0x64
```

Fragmentacja

- można wysłać część danych, nie czekając na całość
  - wszystkie oprócz ostatniej ramki mają `FIN=0`
  - opkode ukazuje się tylko w pierwszej, reszta powinna mieć `0x0`

Ping/Pong

- diagnozowanie połączenia
  - żeby sprawdzić połączenie wysyła się ramkę ping z dowolnym komunikatem
  - w odpowiedzi powinna przyjść ramka pong z tym samym komuikatem
  - „wbudowano w przeglądarkę”, nie ma dostępu z poziomu JavaScriptu
  - serwer zawsze wie, czy klient jest dostępny

Zamykanie połączenia

- przy zamykaniu wysyła się ramkę z kodem operacji `0x8`
- w komunikacie ramki wskazuje się przyczynę zamykania połaczenia
  - w przeglądarce przyczyna będzie dostępna w polu `reason` zdarzenia `close`
- takie zamykanie nazywa się *czystym (clean)*, w polu `wasClean` zdarzenia będzie zapisane `true`

Zamykanie połączenia

- kody zamykania połączenia dostępne są w polu `code` zdarzenia
  - **1000** zamykanie normalne
  - **1001** przeciwny bok „zniknął”
  - **1002** zamykanie w związku z błędem w protokole
  - **1003** zamykanie w związku z otrzymaniem danych, których nie można opracować

Atak „zatruty cache”

- Cache poisoning
  - nawiązanie połączenie przez serwer proxy, który nie wie o WebSocket
  - wysyłanie z klienta po protokole WebSocket ramki, którą serwer proxy zidentyfikuje jako HTTP zapytanie GET (na przykład do http://code.jquery.com/jquery.js)
  - wysyłanie odpowiedzi ze swoim kodem i chache’ującym nagłówkiem
- rozwiązanie: maskowanie

Maskowanie

- klucz maski — losowa 32-bitowa liczba
- generowana przez przeglądarkę, nie znana z góry
- nakłada się na komunikat za pomocą `XOR`
- $x \,\texttt{XOR}\, m \,\texttt{XOR}\, m = x$
- maskowany komunikat jest interpretowany przez proxy jako dane binarne

Przykład. Czat. Kod html

```
 <form name="publish">
  <input type="text" name="message"/>
  <input type="submit"/>
</form>

<div id="subscribe"></div>
```

Przykład. Czat. Klient

```
// połączenie
var socket = new WebSocket("ws://localhost:8081");

// wysłać komunikat z formularza publish
document.forms.publish.onsubmit = function() {
  var outgoingMessage = this.message.value;

  socket.send(outgoingMessage);
  return false;
};
```

Przykład. Czat. Klient, cd

```
// otrzymano komunikat
socket.onmessage = function(event) {
  var incomingMessage = event.data;
  showMessage(incomingMessage);
};

// pokazać komunikat w div#subscribe
function showMessage(message) {
  var messageElem = document.createElement('div');
  messageElem.appendChild(document.createTextNode(message));
  document.getElementById('subscribe').appendChild(messageElem);
}
```

Przykład. Serwer (Node.js)

```
var WebSocketServer = new require('ws');

// połączone klienty
var clients = {};

// WebSocket-    na porcie 8081
var webSocketServer =
    new WebSocketServer.Server({port: 8081});
webSocketServer.on('connection', function(ws) {

  var id = Math.random();
  clients[id] = ws;
  console.log("nowe połączenie " + id);
```

Przykład. Serwer, cd

```
  ws.on('message', function(message) {
    console.log('otzrymano komunikat ' + message);

    for(var key in clients) {
      clients[key].send(message);
    }
  });

  ws.on('close', function() {
    console.log('połączenie zamknięte ' + id);
    delete clients[id];
  });

});
```

Przykład. Uwaga

- potrzebne są dwa dodatkowe moduły: `node-static` oraz `ws`
- instalacja:

```
npm install node-static && npm install ws
```