WebSockets

Szolgáltatásorientált rendszerintegráció

Service-Oriented System Integration

Dr. Balázs Simon

BME, IIT

WebSocket protocol

Original web philosophy

- Client-server model
- Half-duplex:
  - initiator is always the client
  - the client waits for the response of the server
- No server initiation:
  - the client has to poll the server
- HTTP headers for metadata and context information

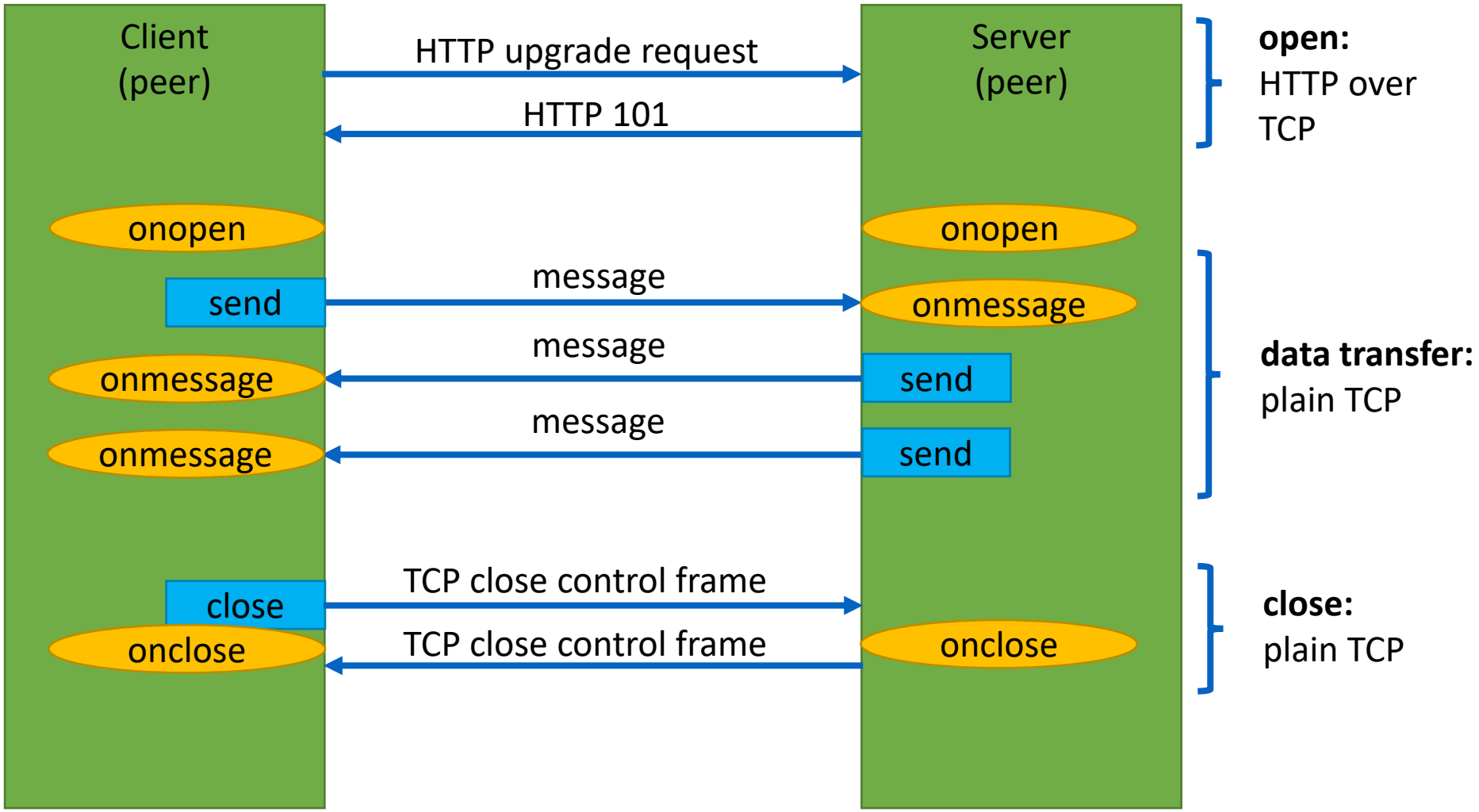
WebSocket protocol

- A single unique plain TCP connection between the two peers
  - HTTP: each request-response requires a new TCP connection
- No HTTP header overhead
- Bidirectional
  - the client can send messages to the server
  - the server can send messages to the client
- Full-duplex:
  - each peer can send multiple messages to the other peer without waiting for a response

WebSocket protocol

- Protocol URI scheme: ws:// or wss://
- Opening a WebSocket connection:
  - an HTTP upgrade request (handshake)
- The same underlying TCP connection used for the HTTP upgrade request will be used for WebSocket communication
- But after the upgrade:
  - no more HTTP headers
  - just plain TCP
  - the application can decide what to send through the TCP connection
- Closing a WebSocket connection:
  - just simply closing the TCP connection

WebSocket protocol



HTTP upgrade request

ws://echo.websocket.org/echo

```
GET /echo HTTP/1.1
Host: echo.websocket.org
User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/36.0
Accept: text/html, application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Sec-WebSocket-Version: 13
Origin: https://www.websocket.org
Sec-WebSocket-Key: gIcmfo3+pI2x3W4i6uT+ig==
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
```

HTTP upgrade response

```
HTTP/1.1 101 Web Socket Protocol Handshake
access-control-allow-credentials: true
access-control-allow-headers: content-type,
    authorization, x-websocket-extensions,
    x-websocket-version, x-websocket-protocol
access-control-allow-origin: https://www.websocket.org
Connection: Upgrade
Date: Wed, 18 Mar 2015 10:35:35 GMT
Server: Kaazing Gateway
Sec-Websocket-Accept: 8XV19zYSfbKMh+ZnY8LkmDrJKpY=
Upgrade: websocket
```

Data transfer

- When the HTTP handshake is over
- It is no longer possible to use HTTP communication
- Any peer (client or server) can send any number of messages
- Data transfer is plain TCP
- Data format: can be text or binary
  - standard subprotocol: e.g. SOAP
  - custom subprotocol: application-specific protocol

.NET APIs for WebSocket

.NET APIs for WebSockets

- Similar interface for the server and the client:
  - Server: WebSocket API for ASP.NET Core
  - Client: ClientWebSocket class
- Very low level API
- Have to write everything manually:
  - opening a connection
  - receiving a message
  - decoding from a byte array
  - encoding to a byte array
  - sending a message
  - closing a connection
  - maintaining a list of clients

Registering a ServerEndpoint service and a WebSocket middleware

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddSingleton<ServerEndpoint>();

var app = builder.Build();
app.UseWebSockets();
app.UseMiddleware<ServerMiddleware>();

app.Run();
```

Server endpoint

```
public class ServerEndpoint
{
  public async Task OnOpen(WebSocket socket)
  {
    Console.WriteLine($"WebSocket opened.");
  }

  public async Task<string?> OnMessage(WebSocket socket, string message)
  {
    Console.WriteLine($"Received: {message}");
    return $"Hello: {message}";
  }

  public async Task OnClose(WebSocket socket)
  {
    Console.WriteLine($"WebSocket closed.");
  }
}
```

WebSocket middleware

```
public class ServerMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ServerEndpoint _server;

    public ServerMiddleware(RequestDelegate next, ServerEndpoint server)
    {
        _next = next;
        _server = server;
    }

    public async Task Invoke(HttpContext context)
    {
        if (!context.WebSockets.IsWebSocketRequest) return;

        var socket = await context.WebSockets.AcceptWebSocketAsync();
        await _server.OnOpen(socket);

        // ...
    }
}
```

WebSocket middleware: Invoke

```
public async Task Invoke(HttpContext context)
{
    if (!context.WebSockets.IsWebSocketRequest) return;

    var socket = await context.WebSockets.AcceptWebSocketAsync();
    await _server.OnOpen(socket);

    try
    {
        while (socket.State == WebSocketState.Open)
        {
            await HandleMessage(socket);
        }
    }
    catch (Exception ex)
    {
        await socket.CloseAsync(WebSocketCloseStatus.InternalServerError,
                                ex.Message, CancellationToken.None);
        throw;
    }
}
```

WebSocket middleware: HandleMessage

```
private async Task HandleMessage(WebSocket socket)
{
    var request = await StringEncoder.ReceiveAsync(socket);
    if (request.message is not null)
    {
        var response = await _server.OnMessage(socket, request.message);
        if (response is not null)
        {
            await StringEncoder.SendAsync(socket, response);
        }
    }
    else if (request.result.MessageType == WebSocketMessageType.Close)
    {
        await _server.OnClose(socket);
        await socket.CloseAsync(WebSocketCloseStatus.NormalClosure,
                                null, CancellationToken.None);
    }
}
```

StringEncoder: Receive

```
public static class StringEncoder
{
  public static async
     Task<(WebSocketReceiveResult result, string? message)>
  ReceiveAsync(WebSocket socket)
  {
    var buffer = new byte[1024 * 4];

    var result = await socket.ReceiveAsync(
          buffer: new ArraySegment<byte>(buffer),
          cancellationToken: CancellationToken.None);

    if (result.MessageType == WebSocketMessageType.Text)
    {
      var text = Encoding.UTF8.GetString(buffer, 0, result.Count);
      return (result, text);
    }
    return (result, null);
  }
}
```

StringEncoder: Send

```
public static class StringEncoder
{
  public static async Task SendAsync(WebSocket socket,
                                     string message)
  {
    var buffer = new ArraySegment<byte>(
       Encoding.ASCII.GetBytes(message), 0, message.Length);

    await socket.SendAsync(buffer: buffer,
                           messageType: WebSocketMessageType.Text,
                           endOfMessage: true,
                           cancellationToken: CancellationToken.None);
  }
}
```

Client

```
using System.Net.WebSockets;
using WebSocketClient;

using (var socket = new ClientWebSocket())
{
    await socket.ConnectAsync(new Uri("wss://localhost:8080"),
                              CancellationToken.None);
    await StringEncoder.SendAsync(socket, "me");
    var response = await StringEncoder.ReceiveAsync(socket);
    if (response.message is not null)
    {
        Console.WriteLine(response.message);
    }
    else if (response.result.MessageType == WebSocketMessageType.Close)
    {
        await socket.CloseAsync(WebSocketCloseStatus.NormalClosure,
                                string.Empty, CancellationToken.None);
    }
    await socket.CloseAsync(WebSocketCloseStatus.NormalClosure,
                            string.Empty, CancellationToken.None);
}
```