Server-Sent Events

Ramin Jafary

Overview

- **Server-Sent Events (SSE) is a server push technology enabling a client to receive automatic updates from a server via an HTTP connection.**
- **Designed to enhance native, cross-browser streaming through a JavaScript API called EventSource.**
- **EventSource API is standardized as part of HTML5 by the W3C.**
- **On September 1, 2006, the Opera web browser implemented this new experimental technology in a feature called "Server-Sent Events".**

Building a real-time web application

- Short polling (client pull)
- Long polling (client pull)
- Server-Sent Events (server push)
- WebSockets (server push)

Short Polling

- Client makes a request to the server
- Server can respond in two ways:
  - It sends an empty response
  - It sends data object in its body (JSON Object)
- Client repeats the above process after a fixed delay.

Challenges

- Establish a new connection (TCP, SSL, and Data)
- Query for new data, respond in-deterministic
- Closing connection and cleanup resources

Long Polling

- Client makes a request to the server

- Server can respond in two ways:

  - If it has some new data available, it can respond right away.

  - If it doesn't have anything new data, it will keep that connection open for a period of time and when it receives new data it will respond back with updated data.

Challenges

- Performance and scaling (resource-intensive)

- Data loss

- Message ordering and delivery when opening multiple connections

WebSockets

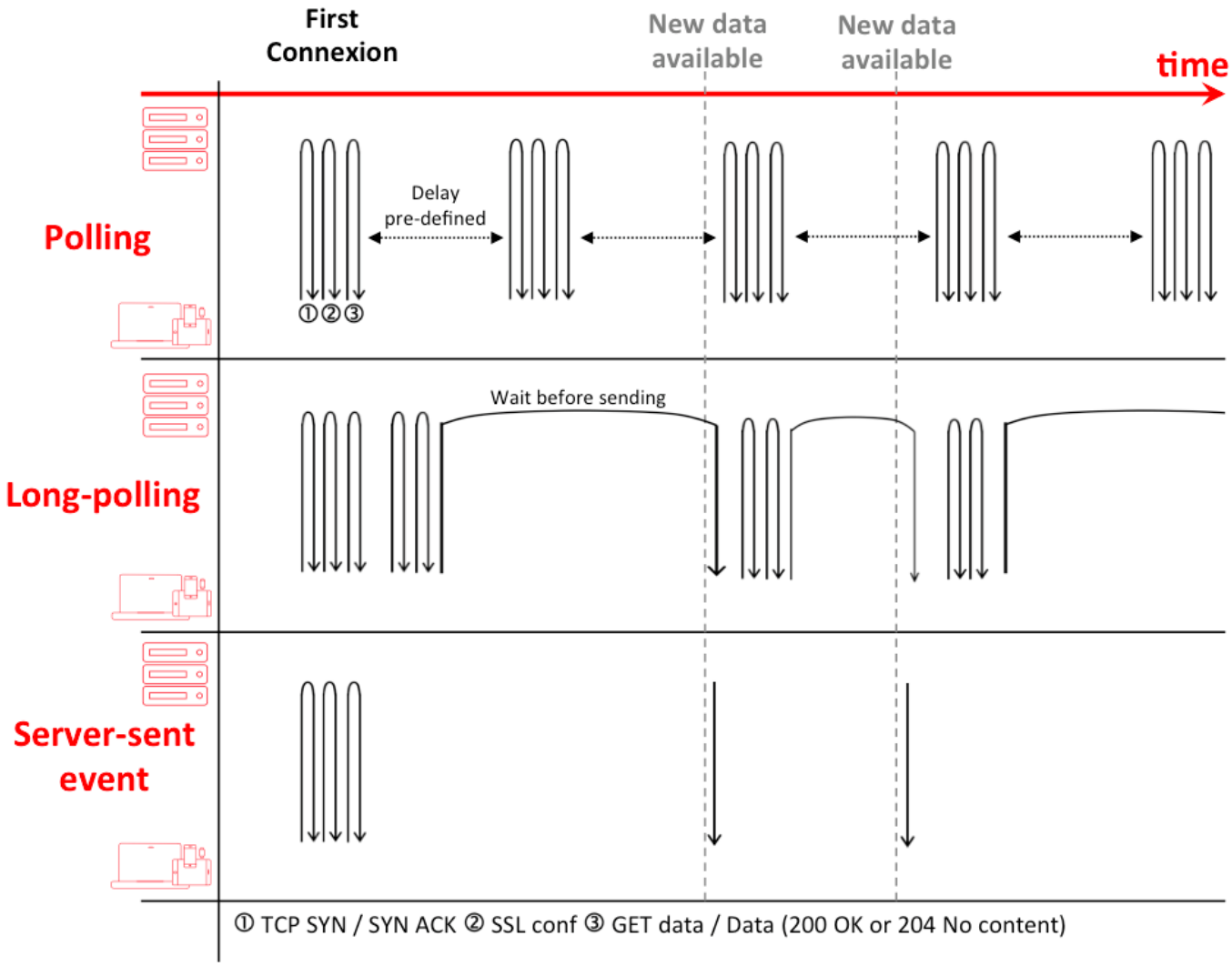
- Real-time data transfer *over* ***a single TCP connection***
- Keeps a unique connection open, allowing messages to be passed back and forth
- Headers are not sent every-time we need to get more information from the server.
- Origin-based security model
- Lightweight footprint on servers
- Supports UTF-8 encoding and binary data

Challenges

- Does not automatically recover when connections are terminated
- Multiplexing over HTTP/2 is complicated
- Use a library to tackle setups and complexities

SSE

- One-way publish-subscribe model (Only Server -> Client data channel)
- Standard JavaScript client API named EventSource
- Multiplexing over HTTP/2 out of the box
- Server can see that the client missed **N** number of messages with unique Id for messages
- Clients automatically handle disconnections by reconnecting
- Only UTF-8 decoding is supported, no binary data
- No way to add headers with the EventSource object
- Max of 6 client connections from a single host (HTTP/1.1)


First Connexion
New data available
New data available
time
Polling
Delay pre-defined
①②③
Long-polling
Wait before sending
Server-sent event
① TCP SYN / SYN ACK ② SSL conf ③ GET data / Data (200 OK or 204 No content)

Generic server implementation

- **Server response headers**

Content-Type: text/event-stream

Cache-Control: no-cache

Connection: keep-alive

- **Body encoding in UTF-8 in the following format**

[field]: value\n

- **Field can have the following values**

data
event
id
retry

: This is a comment ignored by browsers

Response data format

data

data: message\n\n

data: begin message\n
data: continue message\n\n

data: {\n
data: "foo": "bar",\n
data: "baz": 555\n
data: }\n\n

retry

retry: 10000\n

event

event: connected\n
data: User1 just got online\n\n

event: disconnected\n
data: generic unnamed event\n\n

id

id: 1\n
data: message1\n\n

id: X\n
data: messageX\n\n