

Better WebSockets - Server-Sent Events, a carefree alternative

 EuroPython - 12/07/2019 - Andrei Neagu @weethK



About me (can you guess the city?)

Work as an IT Technical Consultant

I like to travel and explore

Also known as “typo master” at work



Schedule

- ◆ SSE introduction
- ◆ Inner workings
- ◆ Differences from WebSockets
- ◆ Implementation explanation for a generic HTTP server in Python
- ◆ Some use cases



Raise hands time 🙋👁️👁️

Server to client data delivery techniques

- ◆ Polling
- ◆ LongPolling
- ◆ WebSockets
- ◆ Server Sent Events



Polling

The dark ages



LongPolling

Slightly less darker



WebSockets

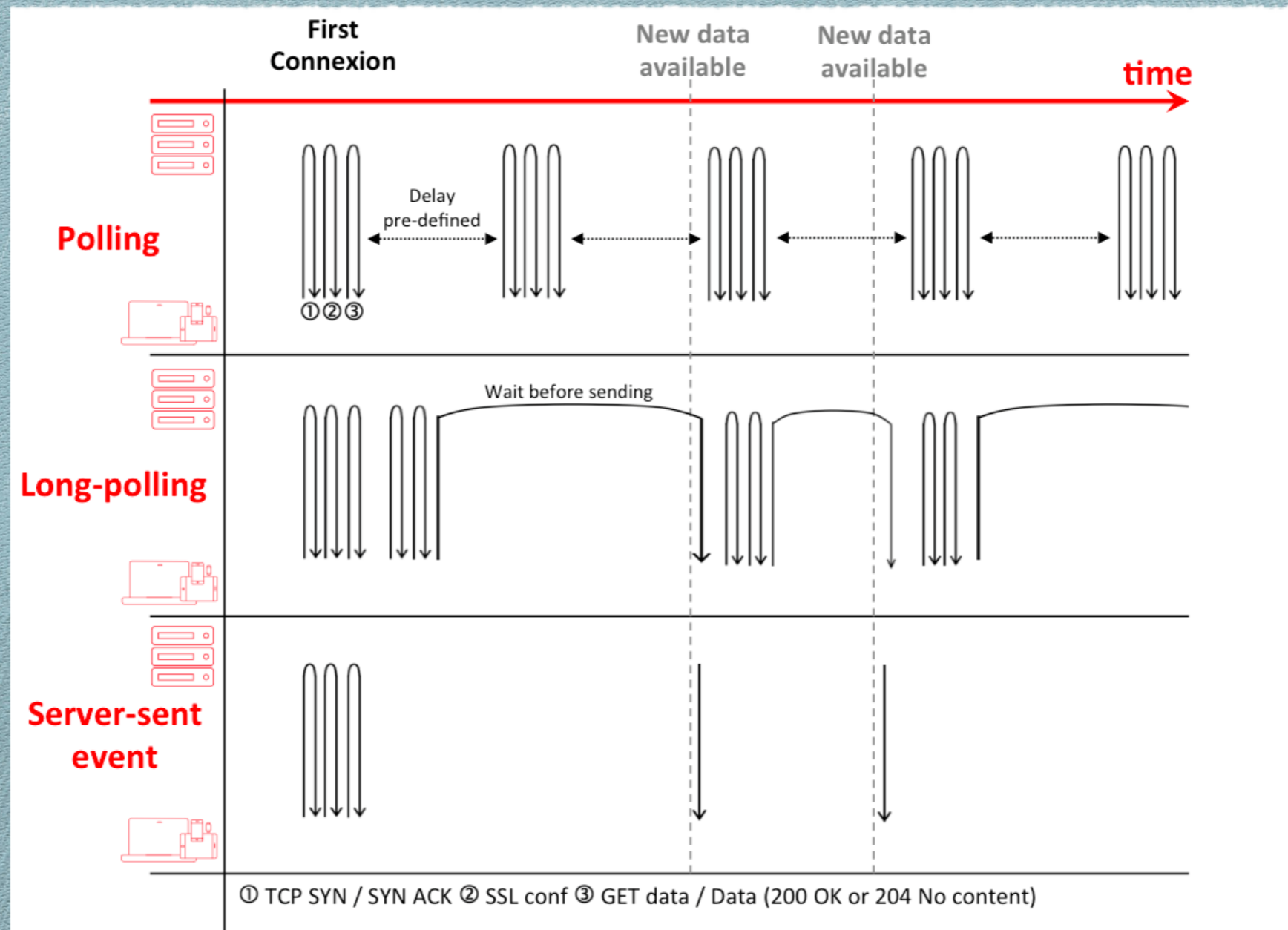
The cool kid, tend to stand out



Server Sent Events

Not that well known

(did you know that a Lavazza museum exist? And that I do not drink coffe?)



Connection wise

Image source: <https://codeburst.io/polling-vs-sse-vs-websocket-how-to-choose-the-right-one-1859e4e13bd9>

The simplest example

Javascript

```
const eventStream = new EventSource( url: "/stream");  
eventStream.addEventListener( type: "message", listener: message =>{  
    // handle message here  
});
```

Python

```
from flask import Flask  
app = Flask(__name__)  
  
# Flask  
@route("/stream")  
def stream():  
    def fetch_data():  
        for message in BLOCKING_DATA_SOURCE:  
            yield 'data: %s\n\n' % message  
  
    return Response(fetch_data(), mimetype="text/event-stream")  
  
if __name__ == '__main__':  
    app.run()
```

More on EventSource

Available handlers

Event handler	Event handler event type
onopen	open
onmessage	message
onerror	error

Usage in JS

```
const eventStream = new EventSource( url: "/stream");
eventStream.addEventListener( type: "message", listener: message => {
    // handle message here
});
eventStream.addEventListener( type: "error", listener: (error) => {
    // error occurred client side
});
eventStream.addEventListener( type: "open", listener: () => {
    // just connected
});

// somewhere else, when you are done with it
eventStream.close();
```

More Python frameworks

```
# TurboGears2
class TheBestController(TGController):
    @expose(content_type='text/event-stream')
    def stream(self, **kwargs):
        def fetch_data():
            for message in BLOCKING_DATA_SOURCE:
                yield 'data: %s\n\n' % message

        return fetch_data()
```

```
# Flask
@route("/stream")
def stream():
    def fetch_data():
        for message in BLOCKING_DATA_SOURCE:
            yield 'data: %s\n\n' % message

    return Response(fetch_data(), mimetype="text/event-stream")
```

```
# Pyramid
@view_config(route_name='events')
def stream(request):
    def fetch_data():
        for message in BLOCKING_DATA_SOURCE:
            yield 'data: %s\n\n' % message

    headers = [('Content-Type', 'text/event-stream'),
               ('Cache-Control', 'no-cache')]
    response = Response(headerlist=headers)
    response.app_iter = fetch_data()
    return response
```

```
# aiohttp
async def stream(request):
    async with sse_response(request) as resp:
        async for message in AWAITABLE_BLOCKING_DATA_SOURCE():
            await resp.send('data: %s\n\n' % message)
    return resp
```

There are some libraries for django

The internals

A brief tour



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: 1° message\n\n

data: 2° begin message\n

data: 2° continue message\n\n

data: {\n

data: "foo": "bar",\n

data: "baz", 555\n

data: }\n\n

event: connected\n

data: User1 just got online\n\n

data: generic unnamed event\n\n

event: disconnected\n

data: User7 abbandona us\n\n

event

id

id: 1\n

data: message1\n\n

id: 2\n

data: message2\n\n

id: X\n

data: messageX\n\n

retry: 10000\n

retry

Custom event listeners example/ client server

Javascript

```
const eventStream = new EventSource( url: "/stream");
eventStream.addEventListener( type: "greet", listener: message => {
  console.log(`Hello ${message.data}`);
});
```

Python

```
from flask import Flask
app = Flask(__name__)

# Flask
@route("/stream")
def stream():
    def fetch_data():
        for name in BLOCKING_DATA_SOURCE:
            yield 'event: greet\ndata: %s\n\n' % name


    return Response(fetch_data(), mimetype="text/event-stream")

if __name__ == '__main__':
    app.run()
```

More on SSE

- ◆ Requests can be redirected HTTP 301(permanent) & 307(temporary)
- ◆ Only UTF-8 decoding is supported, no binary data
- ◆ Protocol supports multiple type of events, default is message
- ◆ Clients always reconnect (no need to handle)
- ◆ Server sends *HTTP 204 No Content* to stop reconnection
- ◆ Limited amount of global connections per site

Server-sent events - LS

Usage % of all users  ?
Global 92.62%

Method of continuously sending data from a server to the browser, rather than repeatedly requesting it (EventSource interface, used to fall under HTML5)

Current aligned	Usage relative	Date relative	Apply filters	Show all	?										
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile*	Chrome for Android	Firefox for Android	IE Mobile	UC Browser for Android	Samsu Interr
		2-5	4-5	3.1-4	10.1	3.2		2.1-4.3							
6-10	12-17	6-66	6-74	5-12	11.5-60	4-12.1		4.4-4.4.4	7	12-12.1			10		4-8
11	18	67	75	12.1	62	12.3	all	67	10	46	75	67	11	11.8	9.2
	76	68-69	76-78	13-TP		13									

Notes Known issues (4) Resources (7) Feedback

MS Edge status: Under Consideration

Native browser support

Source: <https://caniuse.com/#feat=eventsource> 09/07/2019

Other browsers, via **polyfill** <https://github.com/Yaffle/EventSource>

Can I use it without a
browser?

Yes, there are libraries

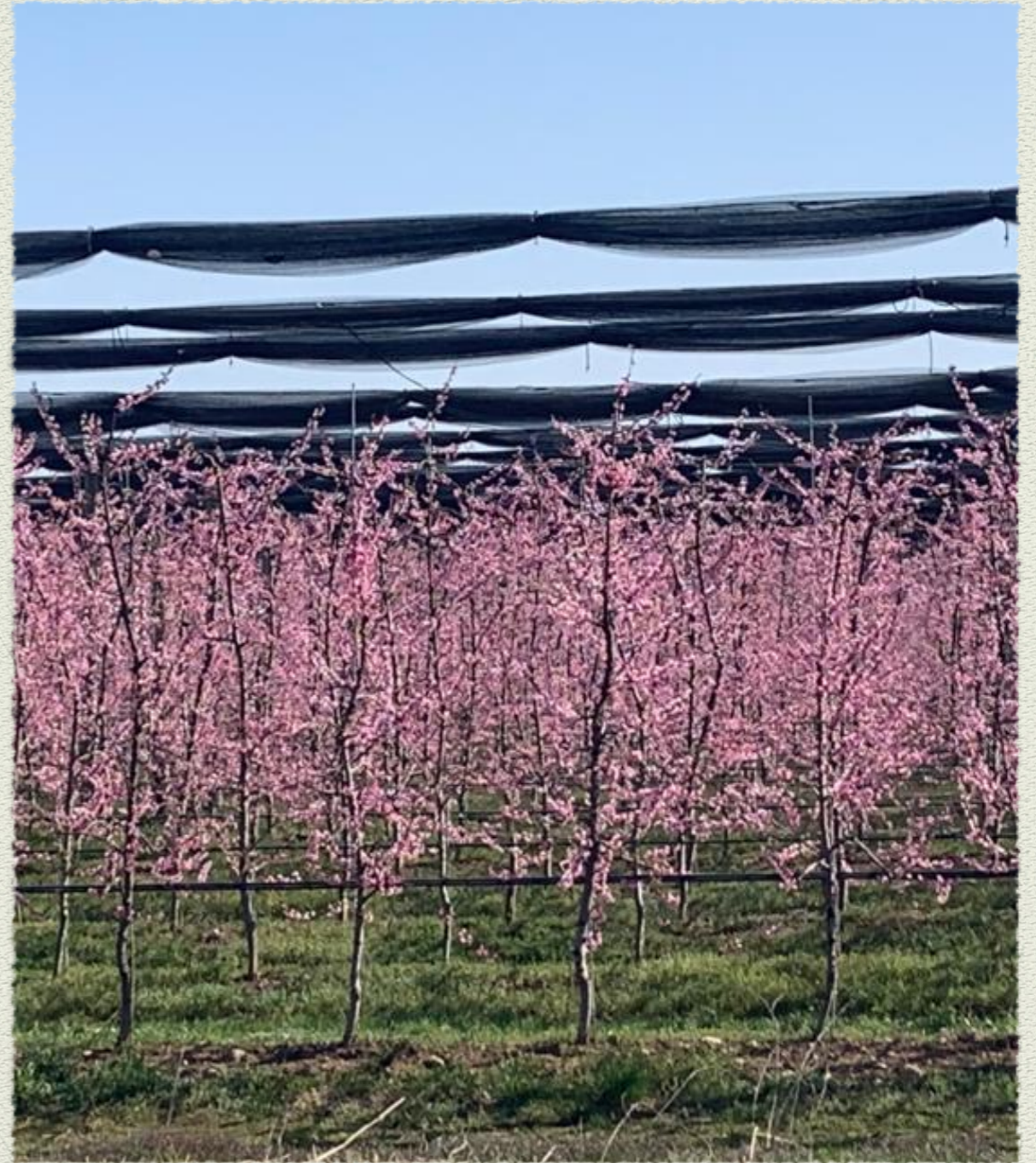
- ◆ Python: sseclient, sseclient-py, aiosseclient
- ◆ Android: sse-android, RxSSE
- ◆ iOS: EventSource(Swift), ios-eventsource(Objective-C)
- ◆ react-native: react-native-event-source (based on a polyfill)

SSE vs WebSockets

- ◆ Only UTF-8 encoding
 - ◆ Uses HTTP
 - ◆ Proxy friendly
 - ◆ Builtin support for reconnection and synchronization
 - ◆ Detects disconnection server side when trying to send out data
 - ◆ Only Server -> Client data channel
 - ◆ Clients automatically handle disconnections by reconnecting
- ◆ Also supports binary data
 - ◆ Has a custom protocol
 - ◆ May have to reconfigure some proxies
 - ◆ Heartbeat, does not always work
 - ◆ Can detect disconnections server side
 - ◆ Can send data in both directions
 - ◆ Client disconnections must be explicitly handled

Use cases

- ◆ Dashboards
- ◆ News feeds
- ◆ Notifications to browser
- ◆ Games (depends on the game)



Some possibly useful links

- ◆ <https://www.w3.org/TR/eventsource/>
- ◆ <https://stackoverflow.com/questions/7636165/how-do-server-sent-events-actually-work>
- ◆ <http://html5doctor.com/server-sent-events/>
- ◆ <https://pythonpedia.com/en/tutorial/9100/python-server-sent-events>
- ◆ <https://streamdata.io/blog/push-sse-vs-websockets/>
- ◆ <https://www.tutorialdocs.com/article/server-sent-events-tutorial.html>

Takeaways

- Consider SSE for your next project
- Choose between SSE and WebSockets as it makes sense for your application

Thank you! Questions?

Contact me on Twitter @weetHK

All the pictures used in this presentation are places from or near Turin