Realtime Collaboration on Web

C.C. @ YoMo

https://github.com/yomorun/yomo

Nov 25 2021
YoMo

https://github.com/yomorun/yomo
<table>
<thead>
<tr>
<th>Data</th>
<th>Len...</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Message</td>
<td>49 B</td>
<td>00:36:25.465</td>
</tr>
<tr>
<td>Binary Message</td>
<td>38 B</td>
<td>00:36:25.466</td>
</tr>
<tr>
<td>Binary Message</td>
<td>54 B</td>
<td>00:36:25.519</td>
</tr>
<tr>
<td>Binary Message</td>
<td>54 B</td>
<td>00:36:25.574</td>
</tr>
<tr>
<td>Binary Message</td>
<td>54 B</td>
<td>00:36:25.575</td>
</tr>
<tr>
<td>Binary Message</td>
<td>54 B</td>
<td>00:36:25.629</td>
</tr>
<tr>
<td>Binary Message</td>
<td>54 B</td>
<td>00:36:25.684</td>
</tr>
<tr>
<td>Binary Message</td>
<td>38 B</td>
<td>00:36:27.188</td>
</tr>
<tr>
<td>Binary Message</td>
<td>37 B</td>
<td>00:36:27.229</td>
</tr>
<tr>
<td>Binary Message</td>
<td>54 B</td>
<td>00:36:27.360</td>
</tr>
<tr>
<td>Binary Message</td>
<td>54 B</td>
<td>00:36:27.415</td>
</tr>
<tr>
<td>Binary Message</td>
<td>54 B</td>
<td>00:36:27.470</td>
</tr>
<tr>
<td>Binary Message</td>
<td>84 B</td>
<td>00:36:28.985</td>
</tr>
<tr>
<td>Binary Message</td>
<td>20 B</td>
<td>00:36:29.047</td>
</tr>
<tr>
<td>Binary Message</td>
<td>10 B</td>
<td>00:36:34.983</td>
</tr>
</tbody>
</table>

```
1 00000000: 6364 6265 6464 6867 6460 6a62 6448 6865
2 00000001: 6000 61fe cfc0 c000 c68c 10cc 7285 tdf9
3 00000002: 7e66 a000 00
4  
```
https://geoping.gg

https://yomo.run
<table>
<thead>
<tr>
<th>Loc</th>
<th>IP</th>
<th>Conn</th>
<th>Proc</th>
<th>TTFB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe</td>
<td>143.204.215.8</td>
<td>1.35ms</td>
<td>153.61ms</td>
<td>173.33ms</td>
</tr>
<tr>
<td>North America</td>
<td>65.8.158.72</td>
<td>0.60ms</td>
<td>31.32ms</td>
<td>57.47ms</td>
</tr>
<tr>
<td>Other Asia</td>
<td>13.33.88.129</td>
<td>0.48ms</td>
<td>501.65ms</td>
<td>511.53ms</td>
</tr>
<tr>
<td>Middle-East</td>
<td>13.227.138.48</td>
<td>1.29ms</td>
<td>222.26ms</td>
<td>246.63ms</td>
</tr>
<tr>
<td>Oceania</td>
<td>65.8.33.90</td>
<td>12.20ms</td>
<td>450.85ms</td>
<td>487.44ms</td>
</tr>
<tr>
<td>South America</td>
<td>52.84.83.91</td>
<td>2.04ms</td>
<td>201.91ms</td>
<td>220.13ms</td>
</tr>
<tr>
<td>Africa</td>
<td>52.85.24.76</td>
<td>0.62ms</td>
<td>280.45ms</td>
<td>484.02ms</td>
</tr>
</tbody>
</table>
Dynamic with...
In the past decades
User - Database
User - User
API & Database, Distributed?
Next.js Conf SE — T-Shirt

$50

Love the new logo

Nice detail!

Should we change this color?

https://yomo.run
Next.js Conf SE — T-Shirt

$50

Love the new logo

Should we change this color?
Web is migrating to the edge
Hydration: React 18, SSR, Streaming

New Suspense SSR Architecture in React 18 #37

gaearon announced in Deep Dive

Overview

React 18 will include architectural improvements to React server-side rendering (SSR) performance. These improvements are substantial and are the culmination of several years of work. Most of these improvements are behind-the-scenes, but there are some opt-in mechanisms you'll want to be aware of, especially if you don't use a framework.

The primary new API is `renderToPipeableStream`, which you can read about in Upgrading to React 18 on the Server. We plan to write more about it in detail as it's not final and there are things to work out.

The primary existing API is `<Suspense>`.

This page is a high-level overview of the new architecture, its design, and the problems it solves.
SSR Streaming

Server-Side Streaming

Concurrent features in React 18 include built-in support for server-side Suspense and SSR streaming support. This allows you to server-render pages using HTTP streaming. This is an experimental feature in Next.js 12, but once enabled, SSR will use the same strict runtime as Middleware.

To enable, use the experimental flag `concurrentFeatures: true`:

```javascript
// next.config.js
module.exports = {
  experimental: {
    concurrentFeatures: true
  }
};
```
The future of edge computing is here
Now you can have the benefits of dynamic at the speed of static with Vercel Edge Functions.

Learn about Edge Functions

Say goodbye to cold boots
With Middleware deployed to Vercel, run globally-deployed functions, without any configuration. Even better? Your Next.js application will be optimized for the best performance every time.

De-risk experimentation
From feature flags to A/B testing, Middleware allows your developers to dynamically reconfigure routing at runtime to remove performance impacts and de-risk experimentation.

Make it personalized and fast
Reduce reliance on client-side JavaScript by deploying on the server without any performance tradeoffs. Your team can now serve users the right experience the first time with just code!
implements react router from hydrogen we get a
Host directly on Shopify

Stay tuned for the upcoming Shopify-powered, global hosting solution for Hydrogen storefronts. Oxygen is the fastest way to deploy Shopify-backed commerce experiences.
Web is migrating to the edge, but realtime still not
## Region-to-Region

<table>
<thead>
<tr>
<th>Source City</th>
<th>Barcelona</th>
<th>Hangzhou</th>
<th>Paris</th>
<th>Shanghai</th>
<th>Tokyo</th>
<th>Toronto</th>
<th>Washington</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amsterdam</td>
<td>31.509ms</td>
<td>209.917ms</td>
<td>11.392ms</td>
<td>197.382ms</td>
<td>231.31ms</td>
<td>81.629ms</td>
<td>82.043ms</td>
</tr>
<tr>
<td>Auckland</td>
<td>368.951ms</td>
<td>505.796ms</td>
<td>303.431ms</td>
<td>407.537ms</td>
<td>190.376ms</td>
<td>259.365ms</td>
<td>250.843ms</td>
</tr>
<tr>
<td>Copenhagen</td>
<td>39.303ms</td>
<td>272.217ms</td>
<td>23.563ms</td>
<td>248.208ms</td>
<td>231.074ms</td>
<td>111.46ms</td>
<td>103.301ms</td>
</tr>
<tr>
<td>Dallas</td>
<td>127.31ms</td>
<td>212.652ms</td>
<td>120.314ms</td>
<td>200.643ms</td>
<td>133.205ms</td>
<td>43.383ms</td>
<td>36.068ms</td>
</tr>
<tr>
<td>Frankfurt</td>
<td>35.154ms</td>
<td>257.19ms</td>
<td>11.317ms</td>
<td>247.415ms</td>
<td>218.633ms</td>
<td>101.32ms</td>
<td>94.13ms</td>
</tr>
<tr>
<td>Hangzhou</td>
<td>N/A</td>
<td>N/A</td>
<td>234.481ms</td>
<td>N/A</td>
<td>447.397ms</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>London</td>
<td>36.727ms</td>
<td>275.612ms</td>
<td>8.448ms</td>
<td>239.426ms</td>
<td>237.192ms</td>
<td>92.216ms</td>
<td>78.112ms</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>154.058ms</td>
<td>154.344ms</td>
<td>159.333ms</td>
<td>162.815ms</td>
<td>116.154ms</td>
<td>70.676ms</td>
<td>67.959ms</td>
</tr>
<tr>
<td>Moscow</td>
<td>82.88ms</td>
<td>225.522ms</td>
<td>50.12ms</td>
<td>203.097ms</td>
<td>264.424ms</td>
<td>131.561ms</td>
<td>126.384ms</td>
</tr>
<tr>
<td>New York</td>
<td>96.042ms</td>
<td>314.683ms</td>
<td>73.104ms</td>
<td>276.22ms</td>
<td>176.782ms</td>
<td>20.472ms</td>
<td>8.942ms</td>
</tr>
<tr>
<td>Paris</td>
<td>27.241ms</td>
<td>284.159ms</td>
<td>—</td>
<td>269.094ms</td>
<td>229.661ms</td>
<td>93.264ms</td>
<td>85.740ms</td>
</tr>
<tr>
<td>Shanghai</td>
<td>N/A</td>
<td>5.886ms</td>
<td>313.506ms</td>
<td>—</td>
<td>N/A</td>
<td>240.598ms</td>
<td>N/A</td>
</tr>
<tr>
<td>Stockholm</td>
<td>51.576ms</td>
<td>215.577ms</td>
<td>20.546ms</td>
<td>246.817ms</td>
<td>238.672ms</td>
<td>98.411ms</td>
<td>100.142ms</td>
</tr>
<tr>
<td>Tokyo</td>
<td>212.995ms</td>
<td>80.695ms</td>
<td>228.448ms</td>
<td>81.588ms</td>
<td>—</td>
<td>163.551ms</td>
<td>172.569ms</td>
</tr>
</tbody>
</table>

Source: [https://yomo.run](https://yomo.run)
YoMo helps organize global workload
New realtime communication protocol
Using WebTransport

WebTransport is an API offering low-latency, bidirectional, client-server messaging. Learn more about its use cases, and how to give feedback about the future of the implementation.

Jun 8, 2020 — Updated Jul 18, 2022

Available in: English, Русский, 中文, 日本語, and 한국어
What's WebTransport?

WebTransport is a web API that uses the HTTP/3 protocol as a bidirectional transport. It's intended for two-way communications between a web client and an HTTP/3 server. It supports sending data both unreliably via its datagram APIs, and reliably via its streams APIs.
Unreliable Support

Datagrams are ideal for sending and receiving data that do not need strong delivery guarantees. Individual packets of data are limited in size by the maximum transmission unit (MTU) of the underlying connection, and may or may not be transmitted successfully, and if they are transferred, they may arrive in an arbitrary order. These characteristics make the datagram APIs ideal for low-latency, best-effort data transmission. You can think of datagrams as user datagram protocol (UDP) messages, but encrypted and congestion-controlled.
Streams API

The streams APIs, in contrast, provide **reliable**, ordered data transfer. They're **well-suited** to scenarios where you need to send or receive one or more streams of ordered data. Using multiple WebTransport streams is analogous to establishing multiple **TCP** connections, but since HTTP/3 uses the lighter-weight **QUIC** protocol under the hood, they can be opened and closed without as much overhead.
Streams API - SendStream

```javascript
// Send two Uint8Arrays to the server.
const stream = await transport.createSendStream();
const writer = stream.writable.getWriter();
const data1 = new Uint8Array([65, 66, 67]);
const data2 = new Uint8Array([68, 69, 70]);
writer.write(data1);
writer.write(data2);
try {
    await writer.close();
    console.log('All data has been sent.');
} catch (error) {
    console.error(`An error occurred: ${error}`);
}
```
Streams API - ReceiveStream

```javascript
async function readFrom(receiveStream) {
  const reader = receiveStream.readable.getReader();
  while (true) {
    const {done, value} = await reader.read();
    if (done) {
      break;
    }
  // value is a Uint8Array
  console.log(value);
  }
}
```

```javascript
const rs = transport.receiveStreams();
const reader = rs.getReader();
while (true) {
  const {done, value} = await reader.read();
  if (done) {
    break;
  }
  // value is an instance of ReceiveStream
  await readFrom(value);
}
```

const rs = transport.receiveBidirectionalStreams();
const reader = rs.getReader();
while (true) {
    const {done, value} = await reader.read();
    if (done) {
        break;
    }
}

// value is a BidirectionalStream
// value.readable is a ReadableStream
// value.writable is a WritableStream
Try it out #

The best way to experiment with WebTransport is to start up a compatible HTTP/3 server. You can then use this page with a basic JavaScript client to try out client/server communications.

Additionally, a community-maintained echo server is available at webtransport.day.
WebTransport over HTTP/3 client

Establish WebTransport connection

URL: https://sgp.echo.webtransport.day

Send data over WebTransport

- Send a datagram
- Open a unidirectional stream
- Open a bidirectional stream

Event log

- Initiating connection...
- Connection ready
- Datagram writer ready
- Datagram reader ready
- Sent datagram: hello Akraino
- Received data: hi there, nextarch.io
- Opened bidirectional stream #1 with data: hi there, nextarch.io
- Received data on stream #1: hi there, nextarch.io

This tool can be used to connect to an arbitrary WebTransport server. It has several limitations:

- It can only send an entirety of a stream at once. Once the stream is opened, all of the data is immediately sent, and the write side of the stream is closed.
- This tool does not listen to server-initiated bidirectional streams.
- Stream IDs are different from the one used by QUIC on the wire, as the on-the-wire IDs are not exposed via the Web API.
- The Webtransport object can be accessed using the developer console.

Learn Resource

- web.dev - Using WebTransport
- w3.org WebTransport
- presence.io
- W3C WebTransport Working Group Updates - October 2021

Try it out

- WebTransport Demo

Serverless

- Write your own
What we are crafting

- **WebTransport-polyfill** works with web browsers other than Chrome
- Provides developers with an easy-to-use **programming interface** rather than standard Streaming API
- Introduces a **wire protocol**, so every front-end developer doesn't need to build their communication specification.
What we are crafting

- **Unified backend** supports both WebTransport and WebSocket.
- Functions can be written in **WebAssembly** and deployed anywhere - browser, edge or cloud.
Thanks

https://github.com/yomorun/yomo

2022 Sep