HTTP/3 и UDP в браузере и Node.js

Об авторе

- Более 10 лет в IT
- Был техлидом на криптобирже, CTO в компании аутсорсере
- Выступал на HolyJS с докладом про победу в GameFi хакатоне
- В свободное время делаю онлайн игры в вебе
- Разрабатывал приложения на протоколах: HTTP, HTTP/3, SPDY, WebSocket, Raw native sockets

Содержание

- Быстрый обзор доступных протоколов в вебе
- Основные нюансы HTTP/3
- Практическое использование WebTransport
- Демо 1 и бенчмарк. Как ведет себя приложение при потере пакетов
- Демо 2. Полноценное приложение, простая онлайн игра
- Выводы

Подборка полезных ссылок 1

- Про WebTransport с HolyJS 2022
  https://holyjs.ru/talks/4bab221ffc52430ba41f3609b1bf35de/
- Прошлое, настоящее и будущее HTTP от Cloudflare
  https://blog.cloudflare.com/http3-the-past-present-and-future/
- Node.js фреймворки для разработки онлайн игр
  - WebRTC - https://geckos.io/
  - WS, WT - https://colyseus.io/

Протоколы доступные в браузере, HTTP 1/2

- Основан на TCP
- Всеми любимые “Ручки”
- Один коннект на запрос в HTTP 1, переиспользуемый в HTTP2
- Опциональный SSL
- Юзкейсы:
  - Более 90% клиент-серверного общения
  - С поллингом почти как вебсокет
  - Статика, файлы и тд

Протоколы доступные в браузере, WebSocket

- Основан на TCP
- Постоянное соединение между клиентом и сервером
- Ниже уровнем, чем http, требуется много дополнительного кода
- Есть обвязки в виде stomp и socket.io
- Опциональный SSL
- Юзкейсы:
  - Чаты и соц.сети
  - Биржи
  - Игры
  - Любой интерактив

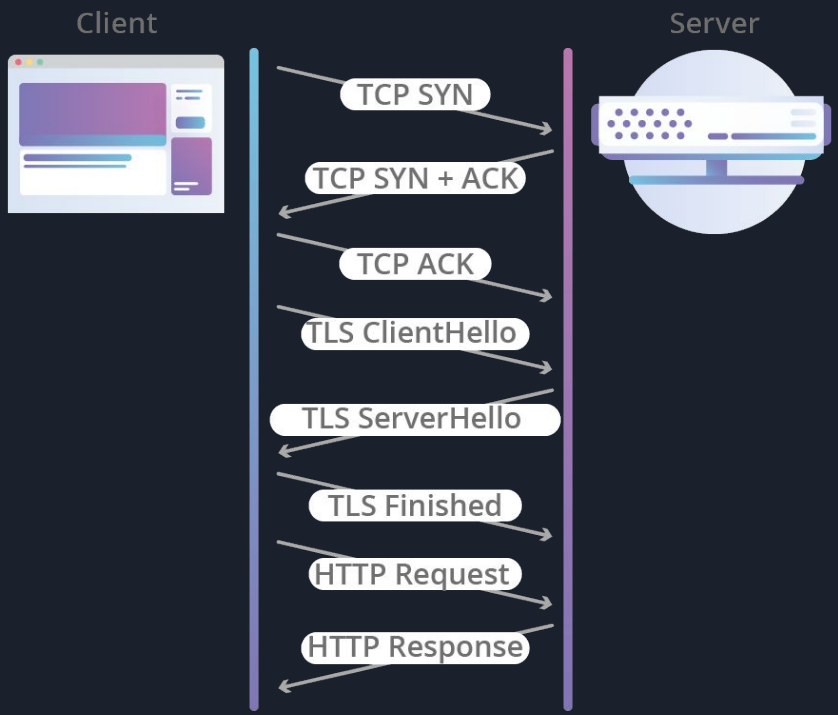
Протоколы доступные в браузере, WebRTC

- Основан на UDP
- P2P протокол
- Низкоуровневый, требуется много дополнительного кода
- Нужен сигнальный сервер для установки соединения
- Юзкейсы:
  - Видео и аудио звонки
  - P2P прямая передача данных

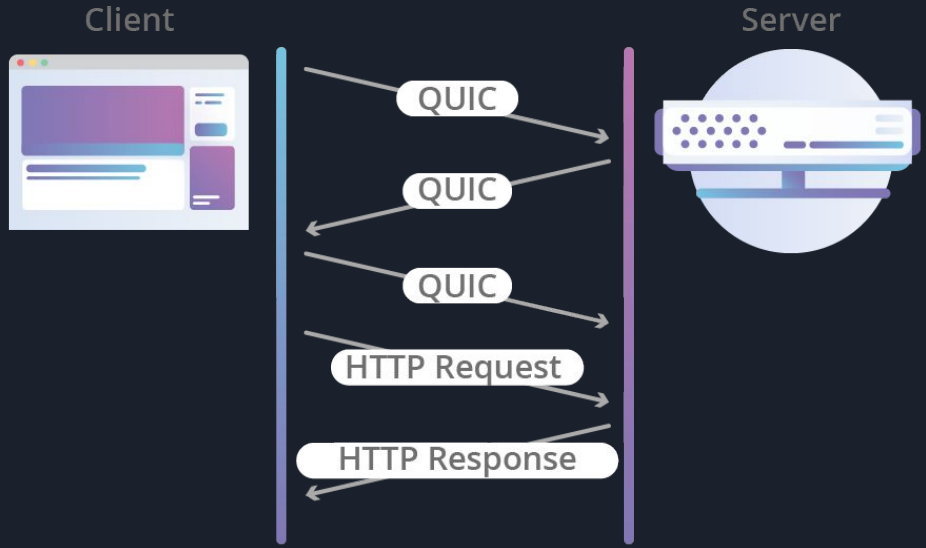
Протоколы доступные в браузере, HTTP/3

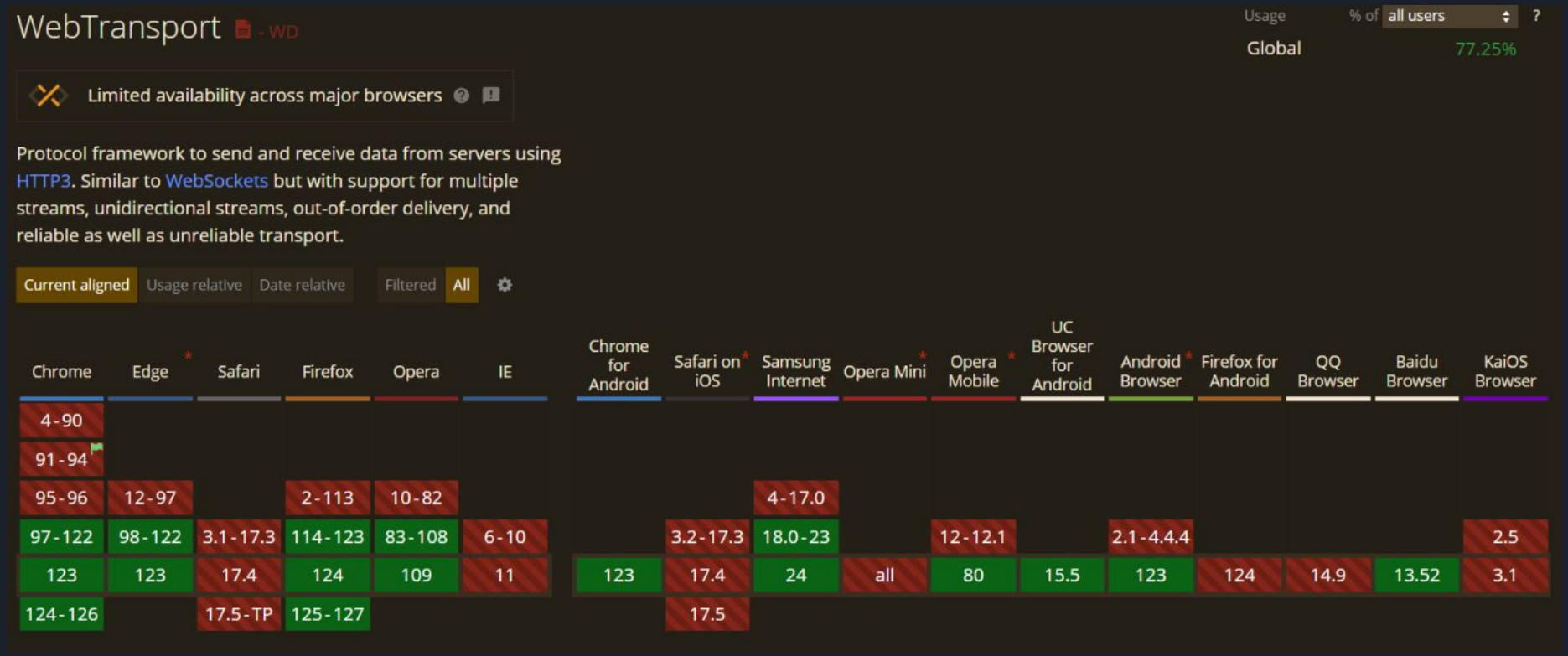
- Основан на QUIC, который реализован поверх UDP
- Обязательный SSL даже для локальной разработки
- Разработчикам доступен новый WebTransport API
- WebTransport очень простой и низкоуровневый, придется писать много кода либо ждать абстракций типа Stomp или Socket.io
- WebTransport заменяет WebSocket, может работать в Reliable и Unreliable режимах
- Обычные HTTP запросы просто поменяют транспорт под капотом, аналогично HTTP/2

HTTP Request Over TCP + TLS



HTTP Request Over QUIC

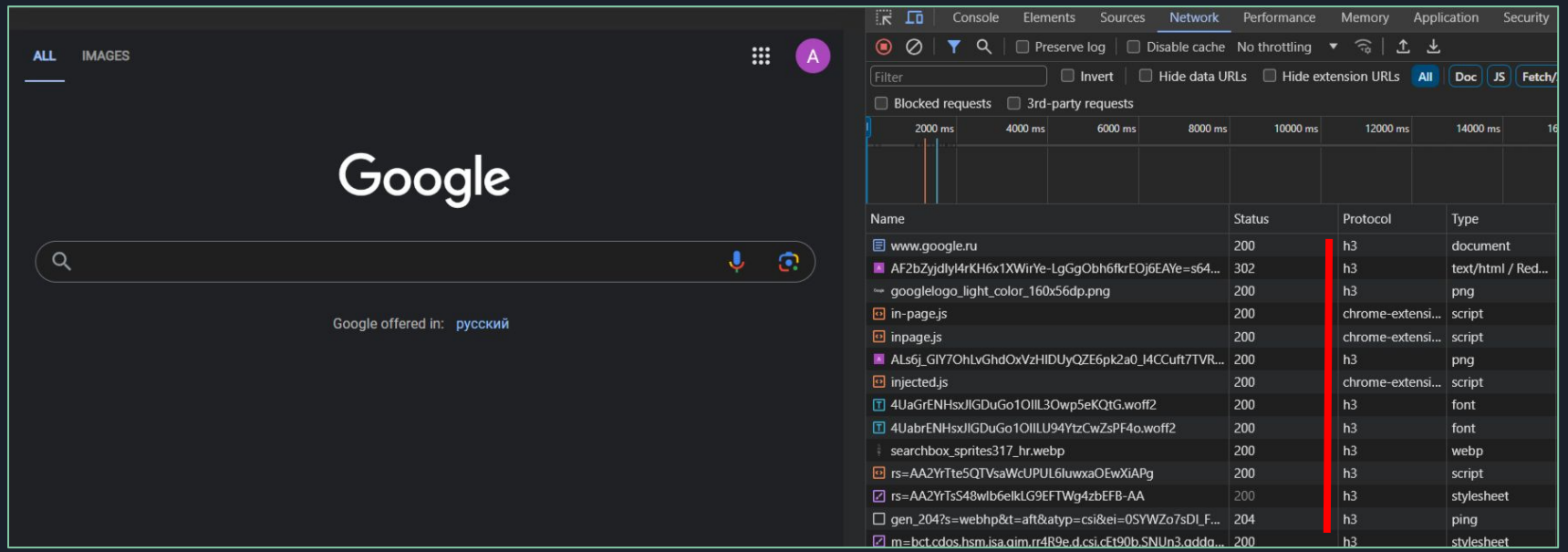




Доступность в браузерах 77.25%

Как проверить работу HTTP/3

- Открыть специальный сайт для клиентской проверки QUIC, например https://cloudflare-quic.com/
- В отладчике убедиться в том, что файлы загружаются по протоколу h3



Нюансы FrontEnd

- В актуальных версиях браузеров уже доступен WebTransport API
- Может понадобится включение дополнительных флагов и параметров браузера
- Для локальной разработки браузер необходимо запускать из терминала с дополнительными аргументами
- Обязателен fallback на более распространенный протокол

```
#!/bin/bash
openssl req -new -x509 -nodes -out cert.pem \
    -keyout key.pem \
    -newkey ec \
    -pkeyopt ec_paramgen_curve:prime256v1 \
    -subj '/CN=127.0.0.1' \
    -days 365
```

```
#!/bin/bash
chrome
    --ignore-certificate-errors-spki-list=xlWN/zo6uNlhvMAgORPO4KLy7Bxp+w8jLUrOvgOc1yk= \
    --origin-to-force-quic-on=23.111.202.19:443 \
    --user-data-dir=quic-user-data \
    https://23.111.202.19:3000
```

Запуск браузера Chrome для локальной разработки

Нюансы BackEnd

- Официально не поддерживается Node, Bun и тд
- Доступно на Node через биндинги quiche от Google

  https://github.com/fails-components/webtransport/

- Есть множество реализаций на других языках https://github.com/google/quiche,

  https://github.com/cloudflare/quiche

Использование WebTransport API

- Streams reader and writer
- Reliable unidirectional stream
- Reliable bidirectional stream
- Unreliable datagrams
- Низкоуровневый бинарный протокол
- Код для клиента и сервера похож, отличается минимально

Подборка полезных ссылок 2

- Документация по WebTrasport от Mozila https://developer.mozilla.org/en-US/docs/Web/API/WebTransport_API
- Примеры и юзкейсы от W3C https://github.com/w3c/webtransport/blob/main/explainer.md
- Пример на Node.js и Socket.io https://socket.io/get-started/webtransport

```
const wt = new WebTransport(`https://${serverAddress}:3000`);
wt.ready.then(() => {
  new WebTransportClient(webTransport);
});

class WebTransportClient {
  constructor(transport) {
    this.datagramsReader = transport.datagrams.readable.getReader();
    this.datagramsWriter = transport.datagrams.writable.getWriter();

    this.bidiStreams = transport.incomingBidirectionalStreams;
    this.bidiReader = undefined;
    this.bidiWriter = undefined;

    this.bidiDataToSend = [];
    this.datagramDataToSend = [];

    this.bidirectionalInit().then(() => {
      this.readData('unreliable');
    });
  }
}
```

```
async bidirectionalInit() {
  const reader = this.bidiStreams.getReader();
  while (true) {
    const { done, value } = await reader.read();
    if (done) {
      break;
    }

    if (!this.bidiReader) {
      this.bidiReader = value.readable.getReader();
      this.bidiReader.closed.catch(e => console.log("Bidi readable closed", e.toString()));
    }

    if (!this.bidiWriter) {
      this.bidiWriter = value.writable.getWriter();
      this.bidiWriter.closed.catch(e => console.log("Bidi writable closed", e.toString()));
    }

    await this.readData('reliable');
  }
}
```

Клиентский код инициализации WebTransport

```
async writeBidiData(data) {
  try {
    if (this.bidiWriter) {
      await this.bidiWriter.write(data);
    }
  } catch (err) {
    console.error('Write bidi data error:', err);
  }
}

async writeDatagramData(data) {
  try {
    if (this.datagramsWriter) {
      await this.datagramsWriter.write(data);
    }
  } catch (err) {
    console.error('Write datagrams data error:', err);
  }
}
```

```
async readData(streamType) {
  const streamReader = streamType === 'unreliable' ? this.datagramsReader : this.bidiReader;

  if (streamReader) {
    let isOpen = true;

    streamReader.closed
      .catch((e) => console.error("Failed to close", e.toString()))
      .finally(() => isOpen = false);

    while (isOpen) {
      try {
        const { done, value } = await streamReader.read();
        if (done) { break; }
        this.routeData(this.binArrayToJson(value), streamType);
      } catch (e) {
        console.error("Failed to read...", e.toString());
        break;
      }
    }
  }
}
```

Клиентский код чтения и записи стримов

Демо 1

Отправка игрового состояния по Unreliable WebTransport и Reliable WebSocket

Как ведут себя оба протокола при потере пакетов

Демо 2

Использование WebTransport на примере онлайн игры в реальном времени

Выводы

- Современные браузеры уже готовы к HTTP/3 и можно делать production-ready приложения
- Данные теперь приходят быстрее и их можно терять
- Юзкейсы unreliable стримов:
  - Быстрые онлайн игры
  - Стриминг аудио и видео
  - Обновление курсов валют на биржах и тд
- Workaround для Node.js
- WebTransport это фактически замена WebSocket
- Протокол низкоуровневый
- https://gitverse.ru/verynimbleferret/HolyJsSpring2024