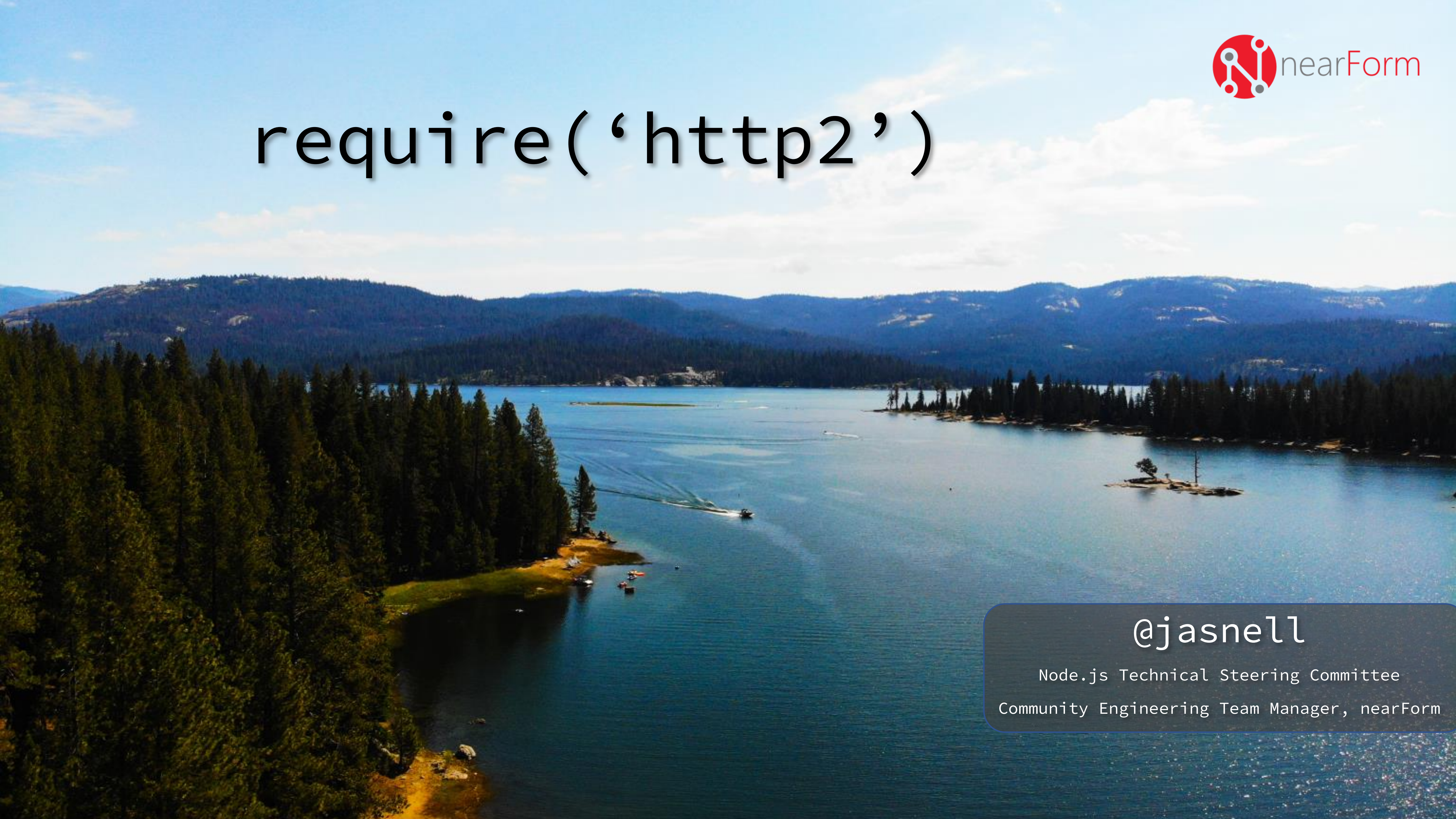


require('http2')

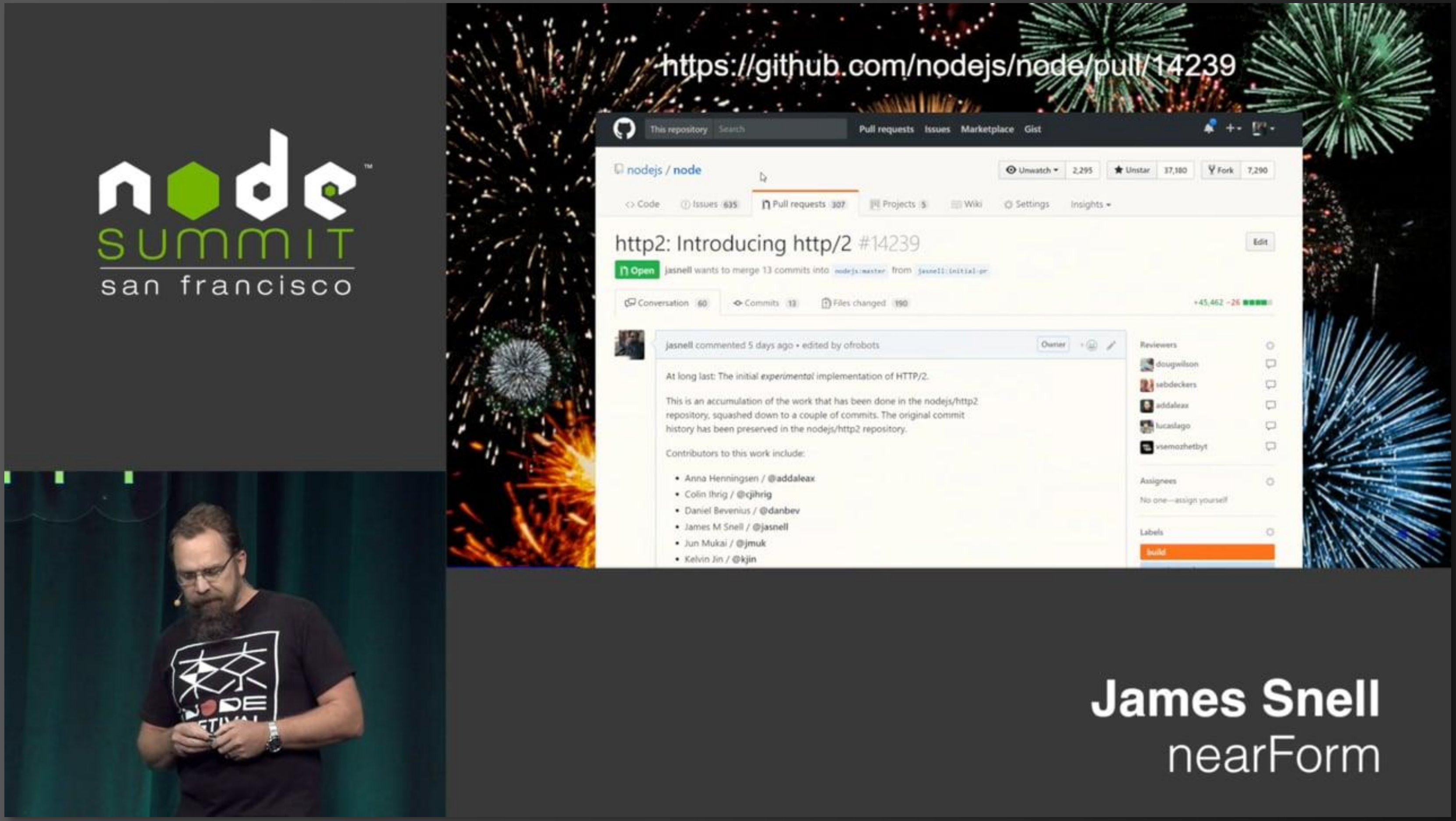
nearForm

@jasnell

Node.js Technical Steering Committee

Community Engineering Team Manager, nearForm

One year ago...



https://github.com/nodejs/node/pull/14239

http2: Introducing http/2 #14239

jasnell wants to merge 13 commits into nodejs:master from jasnell:initial-pr

jasnell commented 5 days ago • edited by ofrobots

At long last: The initial *experimental* implementation of HTTP/2.

This is an accumulation of the work that has been done in the nodejs/http2 repository, squashed down to a couple of commits. The original commit history has been preserved in the nodejs/http2 repository.

Contributors to this work include:

- Anna Henningsen / **@addaleax**
- Colin Ihrig / **@cjihrig**
- Daniel Bevenius / **@danbev**
- James M Snell / **@jasnell**
- Jun Mukai / **@jmuk**
- Kelvin Jin / **@kjin**

James Snell
nearForm

http:// hello world

```
const http2 = require(‘http2’)
const server = http2.createServer()

server.on(‘stream’, (stream, headers) => {
  stream.respond({ ‘:status’: 200 })
  stream.end(‘hello world’)
})

server.listen(8000)
```

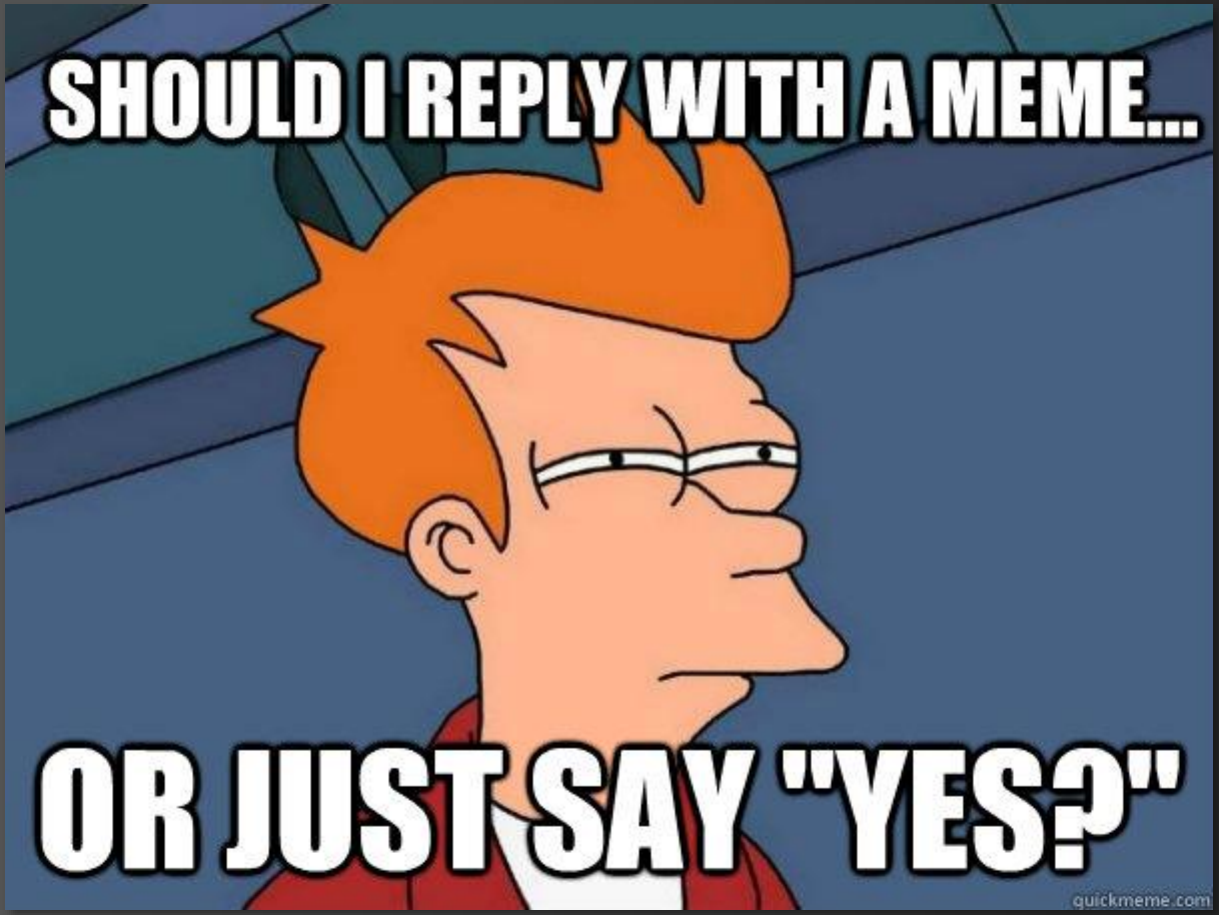
https:// hello world

```
const http2 = require(‘http2’)
const cert = fs.readFileSync(‘my.cert')
const key = fs.readFileSync(‘my.key')
const server = http2.createSecureServer({ cert, key })

server.on(‘stream’, (stream, headers) => {
  stream.respond({ ‘:status’: 200 })
  stream.end(‘hello world’)
})

server.listen(8443)
```

Can you actually deploy and use it?



npm install fastify

```
const fs = require('fs')
const path = require('path')
const fastify = require('fastify')({
  http2: true,
  https: {
    key: fs.readFileSync('fastify.key'),
    cert: fs.readFileSync('fastify.cert')
  }
})

fastify.get('/', function (request, reply) {
  reply.code(200).send({ hello: 'world' })
})

fastify.listen(3000)
```

nearForm

npm install restify

```
const fs = require('fs')
const restify = require('restify')

const srv = restify.createServer({
  http2: {
    cert: fs.readFileSync('my.key'),
    key: fs.readFileSync('my.cert'),
    ca: fs.readFileSync('my.csr')
  }
});

srv.get('/', function(req, res, next) {
    res.send({ hello: 'world' });
    next();
});

srv.listen(3000);
```

restify

npm install hapi

```
const hapi = require('hapi')
const http2 = require('http2’)

const server = hapi.server({
  listener: http2.createSecureServer({ cert: fs.readFileSync(‘my.cert'), key: fs.readFileSync(‘my.key') }),
  port: 8888,
  host: 'localhost'
})

server.route({ method: 'GET', path: '/', handler: (request, h) => 'Hello World!' })

async function init() {
  await server.start()
  console.log(`Server running at ${server.info.uri}`)
}

init()
```

nearForm

But what about the middleware?

NOT SURE IF SITE DOESN'T WORK
OR JUST BECAUSE OF REVERSE PROXY

nginx reverse proxy to http2 server?

nearForm

nghttpx works tho! (https://nghttp2.org)

```
frontend=0.0.0.0,8080
backend=127.0.0.1,8888;/;proto=h2
frontend-no-tls=no
backend-no-tls=no
workers=1
log-level=INFO
private-key-file=/path/to/key.pem
certificate-file=/path/to/cert.pem
```

nghttpx is part of the nghttp2 distribution.

Node.js uses the same nghttp2 library internally.

So does fastify-http-proxy!

```
const fastify = require('fastify')
const server = fastify({ http2: true })

server.register(require('fastify-http-proxy'), {
  upstream: 'http://localhost:8888',
  prefix: '/',
  http2: true
})

server.listen(8080)
```

How’s the performance?

benchmarks

HTTP/1

vs.

HTTP/2

```
const http = require('http')
const fs = require('fs')
const server = http.createServer((request, response) => {
  fs.createReadStream('./alice.html').pipe(response)
})
server.listen(8889)
```

```
const h2 = require('http2')
const server = h2.createServer()
server.on('stream', (stream) => {
  stream.respondWithFile('./alice.html')
})
server.listen(8888)
```

the setup…

h2load –n 20000 –c 8 –t 8 –m 500 http://localhost:8888

Running in an Ubuntu VM on Azure, 16 cores, 128 GB

-n 20000 === 20k Requests
-c 8 === 2 concurrent clients
-t 8 === 2 worker threads
-m 500 === max 500 concurrent requests at a time

Payload is 160k text file.

the results: http/1

```
Application protocol: http/1.1


finished in 7.39s, 2708.05 req/s, 422.33MB/s
traffic: 3.05GB (3270580000) total, 1.47MB (1540000) headers (space savings 0.00%),
3.04GB (3267800000) data
                     min         max         mean         sd        +/- sd
time for request:  469.80ms      1.63s       1.37s    204.36ms    87.92%
time for connect:      60us      191us        87us        43us    87.50%
time to 1st byte:   50.84ms   452.92ms    342.54ms    125.62ms    87.50%
req/s           :    338.63     338.82      338.68        0.07    87.50%
```

the results: http/1

```
Application protocol: http/1.1


finished in 7.39s, 2708.05 req/s, 422.33MB/s
traffic: 3.05GB (3270580000) total, 1.47MB (1540000) headers (space savings 0.00%),
3.04GB (3267800000) data
                     min         max         mean         sd        +/- sd
time for request:  469.80ms       1.63s       1.37s   204.36ms    87.92%
time for connect:      60us       191us        87us       43us    87.50%
time to 1st byte:   50.84ms    452.92ms    342.54ms   125.62ms    87.50%
req/s           :    338.63      338.82      338.68       0.07    87.50%
```

the results: http/2

```
Application protocol: h2c


finished in 3.53s, 5666.36 req/s, 883.58MB/s

traffic: 3.05GB (3270160880) total, 196.03KB (200736) headers (space savings
84.07%), 3.04GB (3267800000) data

                     min         max         mean         sd        +/- sd

time for request:   124.21ms    234.75ms    140.47ms    21.72ms    92.00%

time for connect:       81us       347us       141us       88us    87.50%

time to 1st byte:   133.16ms    146.78ms    143.50ms     4.36ms    87.50%

req/s           :     708.81      712.27      710.90       1.11    75.00%
```

the results: http/2

```
Application protocol: h2c


finished in 3.53s, 5666.36 req/s, 883.58MB/s
traffic: 3.05GB (3270160880) total, 196.03KB (200736) headers (space savings
84.07%), 3.04GB (3267800000) data
                     min         max         mean         sd        +/- sd
time for request:   124.21ms    234.75ms    140.47ms     21.72ms    92.00%
time for connect:       81us       347us       141us        88us    87.50%
time to 1st byte:   133.16ms    146.78ms    143.50ms      4.36ms    87.50%
req/s           :     708.81      712.27      710.90        1.11    75.00%
```

What about WebSockets?

HTTP2 !== WebSockets

HTTP2 does not replace WebSockets,
And technically they're incompatible with one another…

But…

We *CAN* support HTTP/2, HTTP/1, and WebSockets in a single server.

nearForm

tls + http/2 + http/1 + websockets

```
const http2 = require('http2')
const fs = require('fs')
const ws = require('ws')

const server = http2.createSecureServer({ key: /**/, cert: /**/, allowHTTP1: true })
server.on('request', (req, res) => { res.end('ok') })

const wss = new ws.Server({ server })
wss.on('connection', (ws) => {
  ws.on('message', (message) => {
    console.log('received: %s', message)
  })
  ws.send('something')
})

server.listen(8443)
```

tls + http/2 + http/1 + websockets

```
const http2 = require('http2')
const fs = require('fs')
const ws = require('ws')

const server = http2.createSecureServer({ key: /**/, cert: /**/, allowHTTP1: true })
server.on('request', (req, res) => { res.end('ok') })

const wss = new ws.Server({ server })
wss.on('connection', (ws) => {
  ws.on('message', (message) => {
    console.log('received: %s', message)
  })
  ws.send('something')
})

server.listen(8443)
```

Step 1: Create the HTTP2 Server and allow it to accept HTTP1 connections

tls + http/2 + http/1 + websockets

```
const http2 = require('http2')
const fs = require('fs')
const ws = require('ws')

const server = http2.createSecureServer({ key: /**/, cert: /**/, allowHTTP1: true })
server.on('request', (req, res) => { res.end('ok') })

const wss = new ws.Server({ server })
wss.on('connection', (ws) => {
  ws.on('message', (message) => {
    console.log('received: %s', message)
  })
  ws.send('something')
})

server.listen(8443)
```

Step 2: Use the same handler for HTTP/2 and HTTP/1 responses

tls + http/2 + http/1 + websockets

```
const http2 = require('http2')
const fs = require('fs')
const ws = require('ws')

const server = http2.createSecureServer({ key: /**/, cert: /**/, allowHTTP1: true })
server.on('request', (req, res) => { res.end('ok') })

const wss = new ws.Server({ server })
wss.on('connection', (ws) => {
  ws.on('message', (message) => {
    console.log('received: %s', message)
  })
  ws.send('something')
})

server.listen(8443)
```

Step 3: Create a WebSocket server and point it at the HTTP2 server…

Some other fun stuff to try…

Server-sent events!

Server Pushed Streams for non-browser clients!

Creating Proxy Tunnels using CONNECT!

Server-sent events

```
const { createSecureServer } = require('http2')
const pem = require('https-pem')
const { finished } = require('stream')
const server = createSecureServer(pem, (req, res) => {
 if (req.url === '/') {
   res.end(`
<html><script>
   const ev = new EventSource('/time');
   ev.addEventListener('time', (result) => {
     document.getElementById("time").innerHTML += '<li>' +
     result.data + '</li>' })
 </script>
 <body> Hello Server-Sent Events <ul id="time"> </ul> </body>
</html>
`)
   return
 } else if (req.url === '/time') {
   res.setHeader('content-type', 'text/event-stream')
   const interval = setInterval(() => {
     res.write(`event: time\ndata: ${new Date().toISOString()}\n\n`)
   }, 1000)
   finished(res, () => { clearInterval(interval) })
   return
 }
 res.statusCode = 404
 res.end('Not found')
})
server.listen(8082)
```

The Server sent events reuse the same HTTP/2 connection already established with the server.

Server-pushed Streams

```
const http2 = require('http2')
const client = http2.connect('https://myserver')
const req = client.request()

client.on('stream', (stream, requestHeaders) => {
  stream.on('push', (responseHeaders) => {  /* .. */ })
  stream.on('data', (chunk) => {  /* .. */ })
  stream.on('end', () => { /* .. */ })
})
req.resume()
```

With non-browser clients, the Server Pushed Streams feature of HTTP/2 offers an entirely new way of implementing server-sent events

Proxying using CONNECT

```
const net = require('net')
const http2 = require('http2')
const { URL } = require('url')
const { NGHTTP2_CONNECT_ERROR, NGHTTP2_REFUSED_STREAM } = http2.constants
const server = net.createServer((socket) => {
 let data = ''
 socket.setEncoding('utf8')
 socket.on('data', (chunk) => data += chunk)
 socket.end('hello')
})
server.listen(0, () => {
 const port = server.address().port
 const proxy = http2.createServer()
 proxy.on('stream', (stream, headers) => {
  if (headers[':method] !== 'CONNECT') {
   stream.close(NGHTTP2_REFUSED_STREAM)
   return
  }
  const auth = new URL(`tcp://${headers[':authority']}`)
  const socket = net.connect(auth.port, auth.hostname, () => {
   stream.respond()
   socket.pipe(stream)
   stream.pipe(socket)
  })
  socket.on('error', (error) => { stream.close(NGHTTP2_CONNECT_ERROR) })
 })
 proxy.listen(0, () => {
  const client = http2.connect(`http://localhost:${proxy.address().port}`)
  const req = client.request({ ':method': 'CONNECT’, ':authority': `localhost:${port}` })
  let data = ''
  req.setEncoding('utf8')
  req.on('data', (chunk) => data += chunk)
  req.on('end', () => {
   client.close()
   proxy.close()
   server.close()
  })
  req.end('hello')
 })
})
```

Proxying using CONNECT

```
const net = require('net')
const http2 = require('http2')
const { URL } = require('url')
const { NGHTTP2_CONNECT_ERROR, NGHTTP2_REFUSED_STREAM } = http2.constants
const server = net.createServer((socket) => {
 let data = ''
 socket.setEncoding('utf8')
 socket.on('data', (chunk) => data += chunk)
 socket.end('hello')
})
server.listen(0, () => {
 const port = server.address().port
 const proxy = http2.createServer()
 proxy.on('stream', (stream, headers) => {
  if (headers[':method] !== 'CONNECT') {
   stream.close(NGHTTP2_REFUSED_STREAM)
   return
  }
  const auth = new URL(`tcp://${headers[':authority']}`)
  const socket = net.connect(auth.port, auth.hostname, () => {
   stream.respond()
   socket.pipe(stream)
   stream.pipe(socket)
  })
  socket.on('error', (error) => { stream.close(NGHTTP2_CONNECT_ERROR) })
 })
 proxy.listen(0, () => {
  const client = http2.connect(`http://localhost:${proxy.address().port}`)
  const req = client.request({ ':method': 'CONNECT', ':authority': `localhost:${port}` })
  let data = ''
  req.setEncoding('utf8')
  req.on('data', (chunk) => data += chunk)
  req.on('end', () => {
   client.close()
   proxy.close()
   server.close()
  })
  req.end('hello')
 })
})
```

The CONNECT tunnel uses a single HTTP/2 Stream… which means you can open multiple tunnels over a single HTTP/2 Connection.

What about Debugging?

Client:
nghttp2 -v (verbose)

Server:
NODE_DEBUG=http2
NODE_DEBUG_NATIVE=http2
@code and Chrome DevTools Debuggers

Want even more?

node –trace-event-categories node.async_hooks

What's left to do?

- We still have a few bugs to work out
- Enabling detailed http2 trace events
- A better client would be nice

- Mostly it depends on you
  - Build stuff
    - Tell us what works, and what doesn't.
    - Tell us what's useful, and what's not.
  - ***We need feedback from implementers***

(Yes, yes you should)

nearForm

Thank you.

@jasnell
jasnell@nearform.com
jasnell@gmail.com