Microservice interaction with HTTP/2

DotNext 2018 Piter

Zhirov Evgeny, Kontur

Agenda

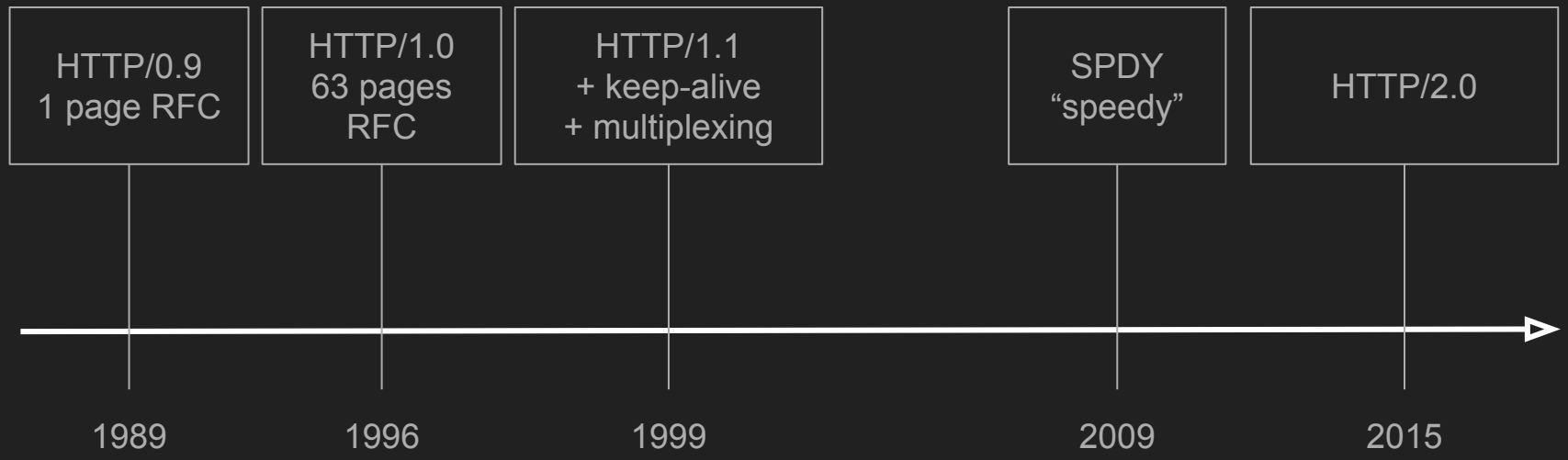
HTTP/2: what’s new?

Long polling scenario

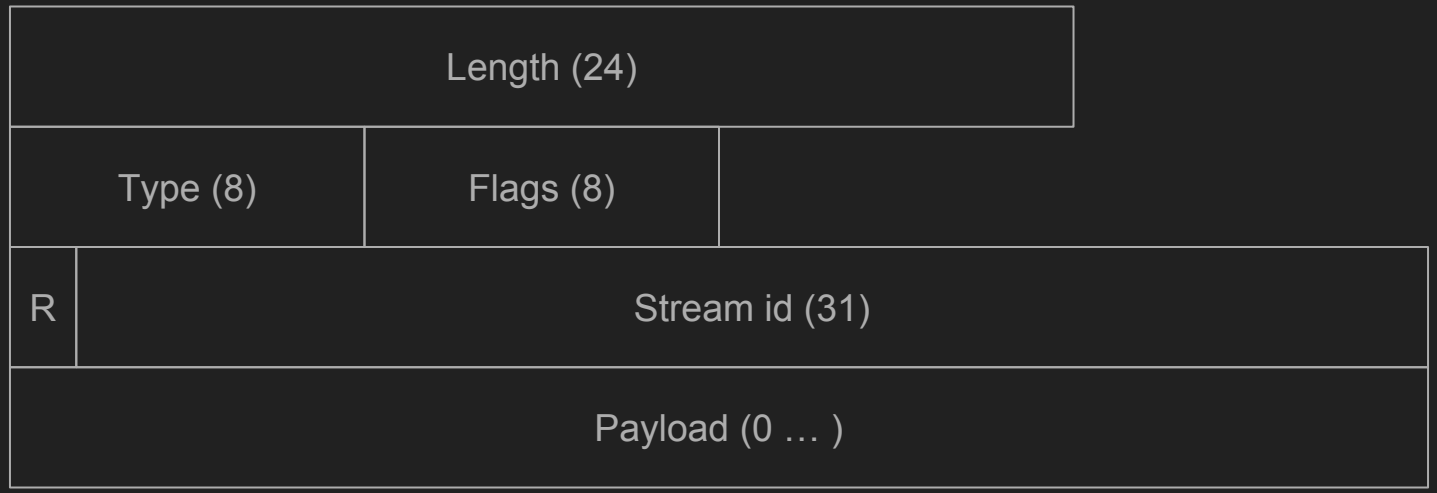
Caveats: Windows and .NET Framework

What about .NET Core and Linux?

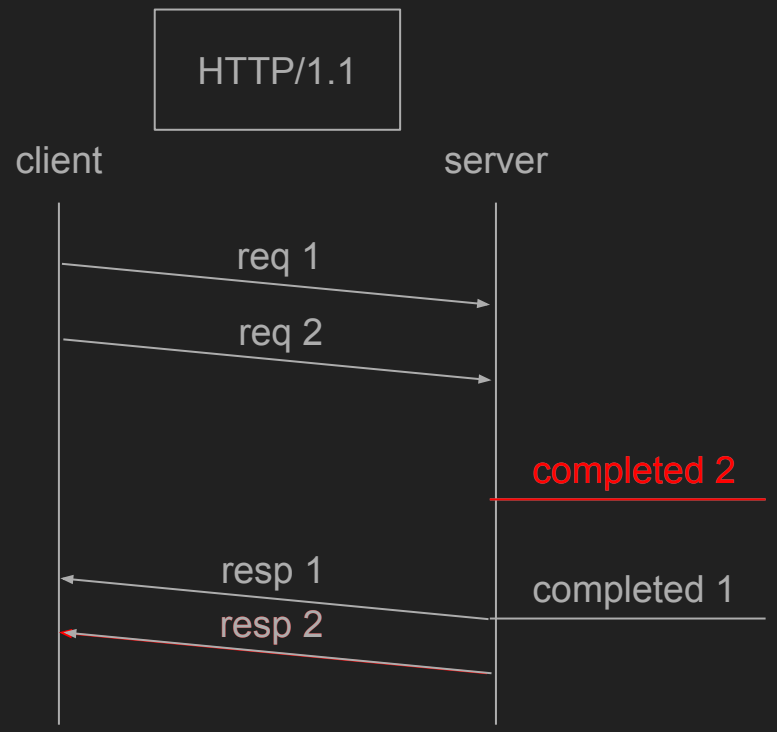
Historic timeline

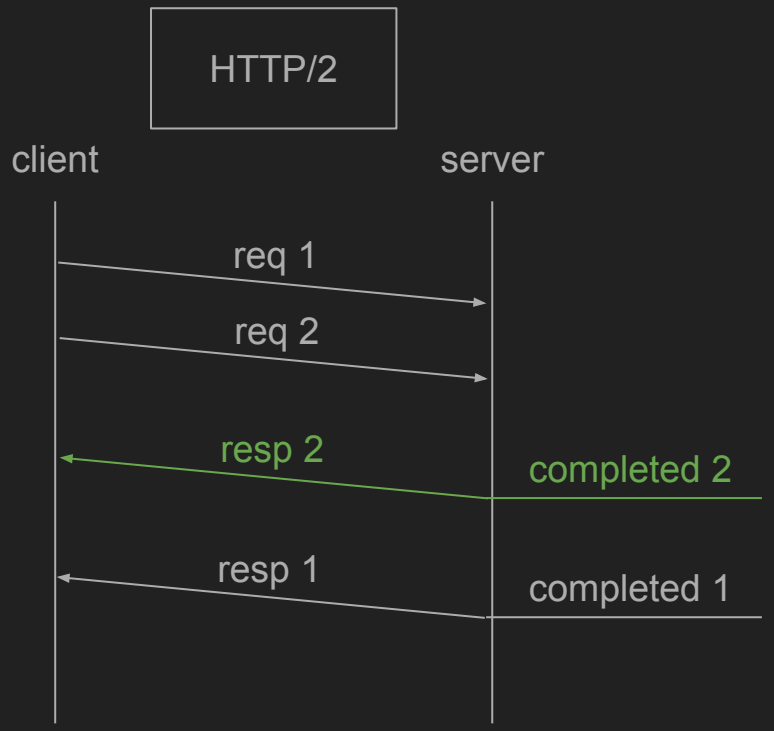


Binary protocol



Multiplexing





And more optimizations for browsers

Multiplexed requests prioritization

Header compression

Server push

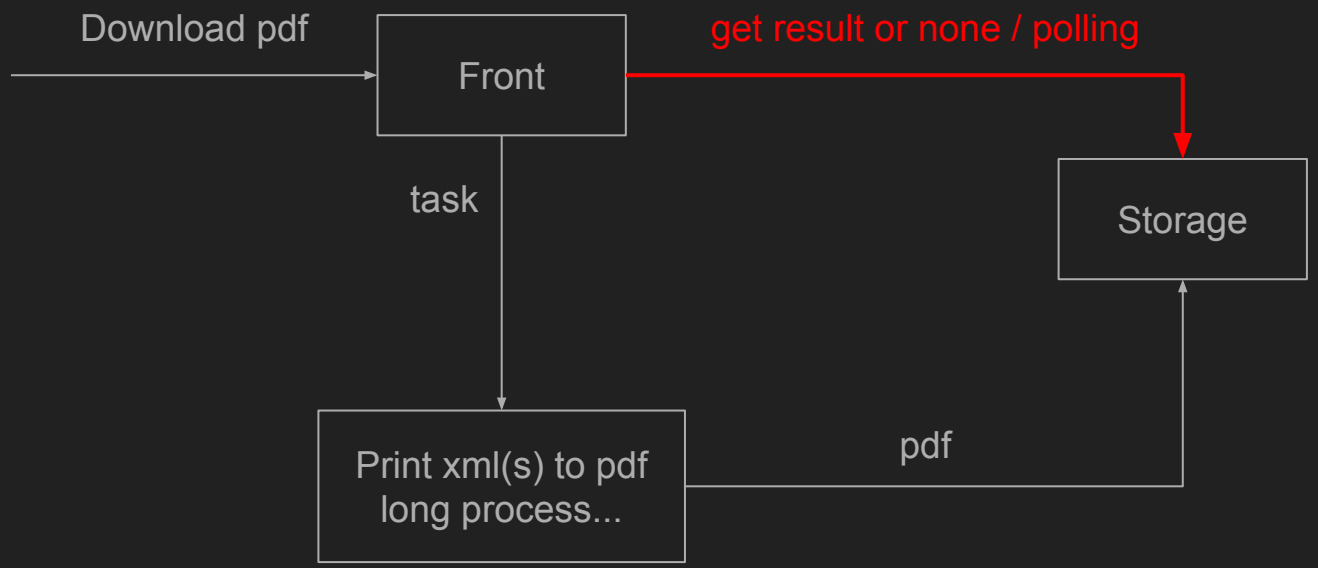
Infrastructure tasks

Common solutions for Kontur teams

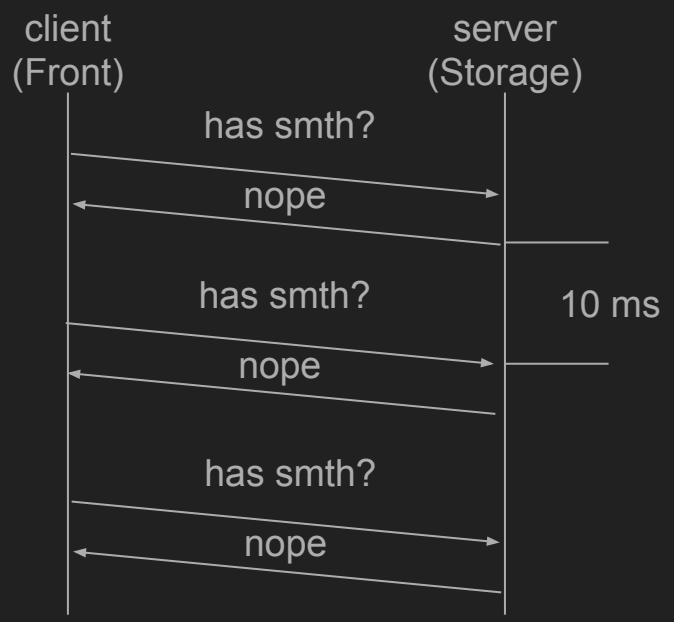
High-performance HTTP API without web front

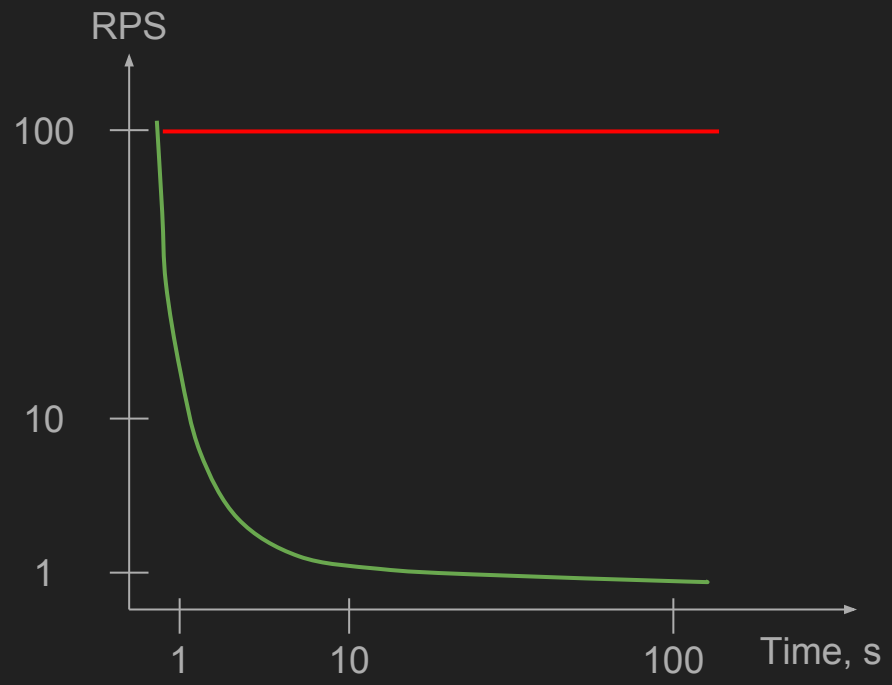
Data storages, task queues, etc.

PDF download

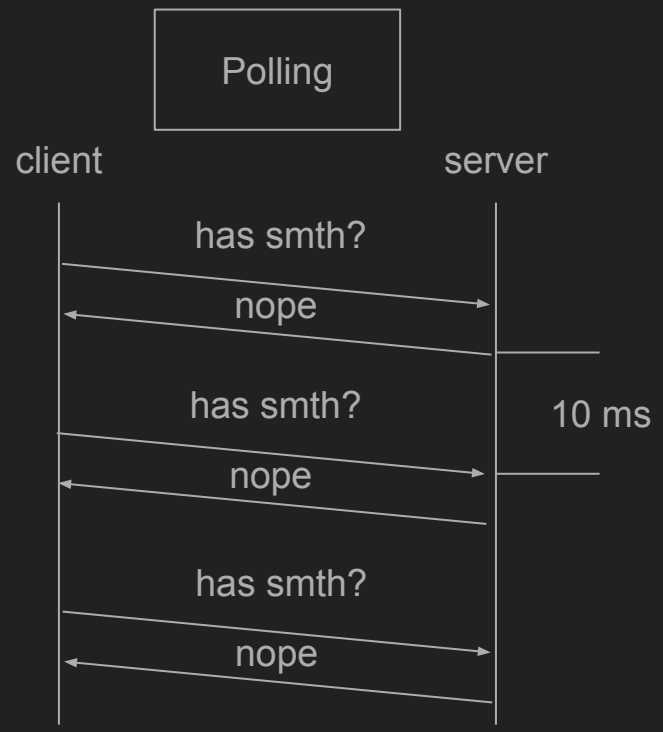


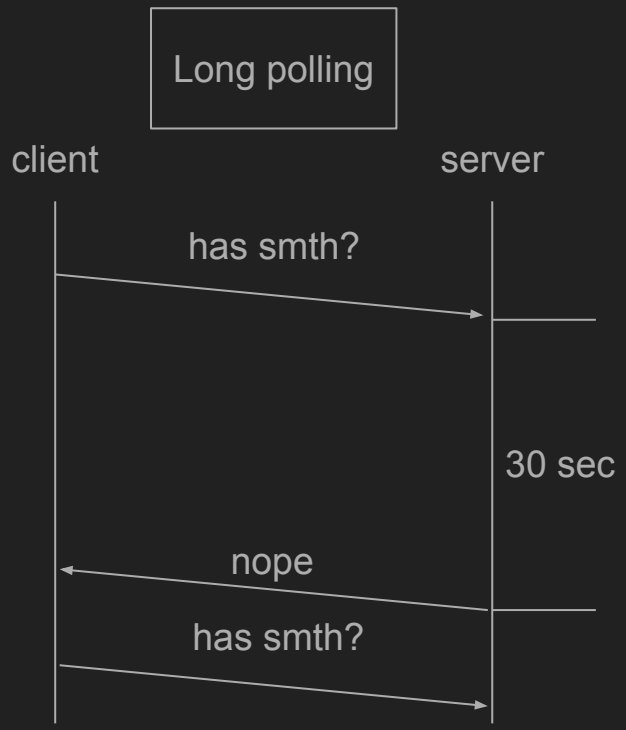
Polling


client
(Front)
server
(Storage)
has smth?
nope
has smth?
nope
has smth?
nope
10 ms


RPS
100
10
1
1
10
100
Time, s

Optimize poll strategy





HTTP/1.1 limitations

Can’t use multiplexing (1st can block others)

One long-poll request = one tcp connection

TCP connection = (client_ip, client_port, server_ip, server_port)

client_port = [0..65535]

Load per front replica

300 “download pdf” RPS — 300 simultaneous tcp connections

Something happened with printing backend

After 1 second — 600 connections

After 1 minute — 18k connections

After 4 minutes — can’t open new tcp connection

Why HTTP/2

WebSockets, TCP, SignalR — completely new stack

No semantics changes in HTTP/2

Can reuse our http infrastructure: distributed tracing, metrics, intelligent replica selection, retry strategies, etc...

Environment

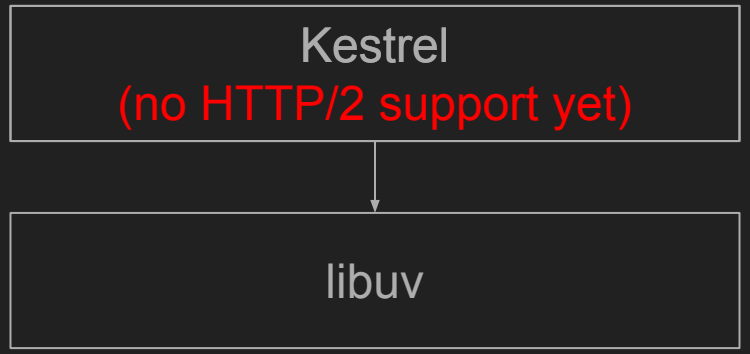
Windows Server 2016 / Windows 10

.NET Framework 4.6.2

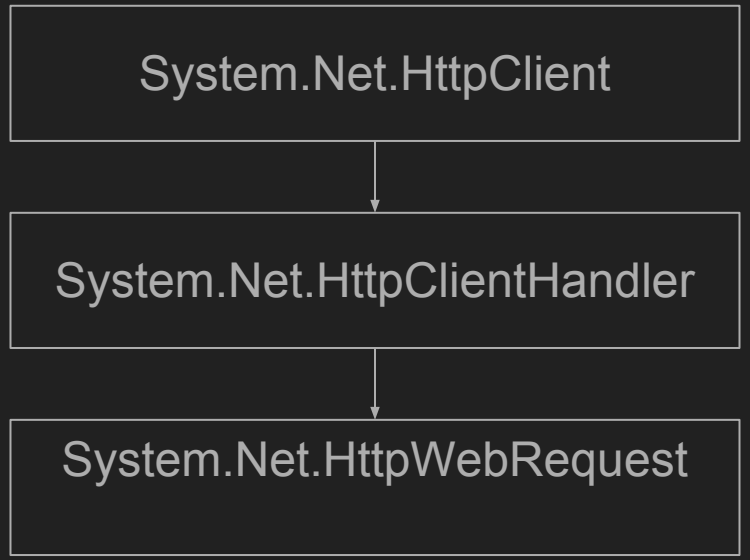
September, 2017

What if <.NET Core, Linux, etc…> — later!

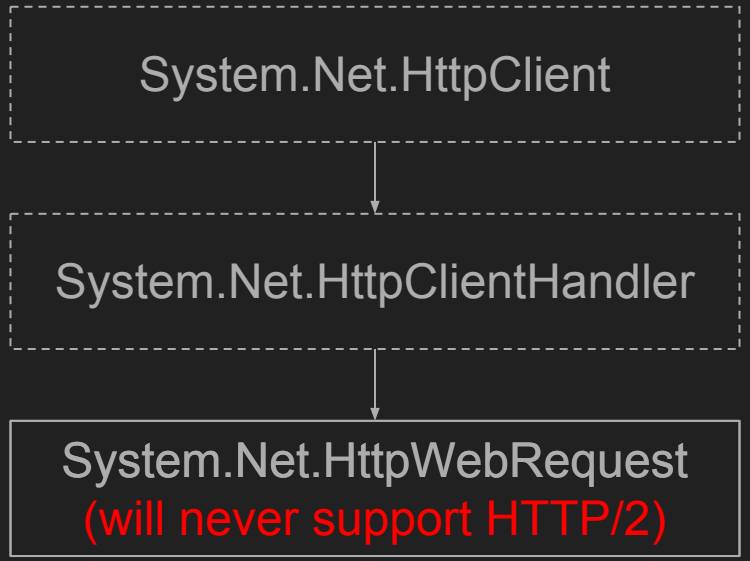
HTTP/1.1 server on .NET Framework



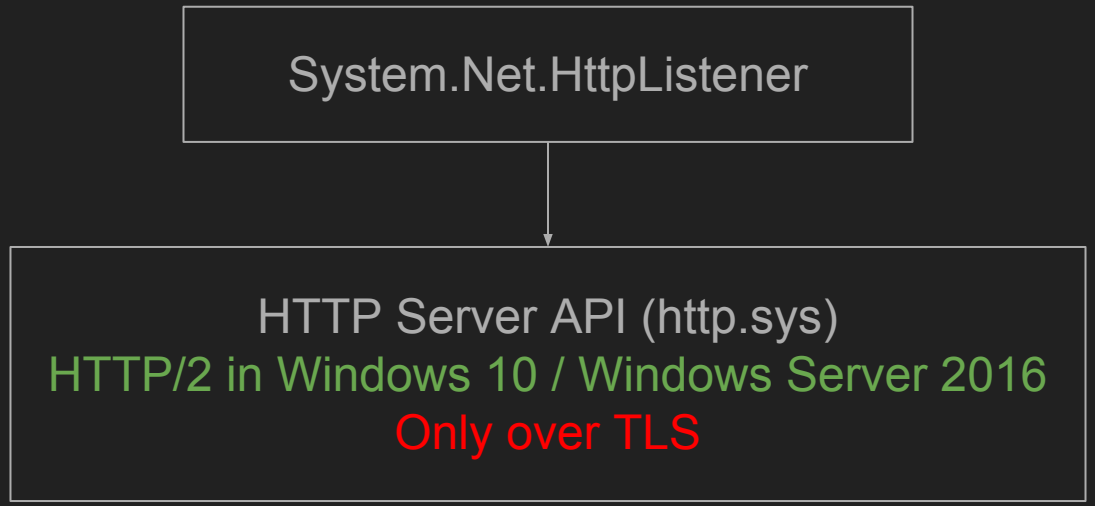
HTTP/1.1 client on .NET Framework



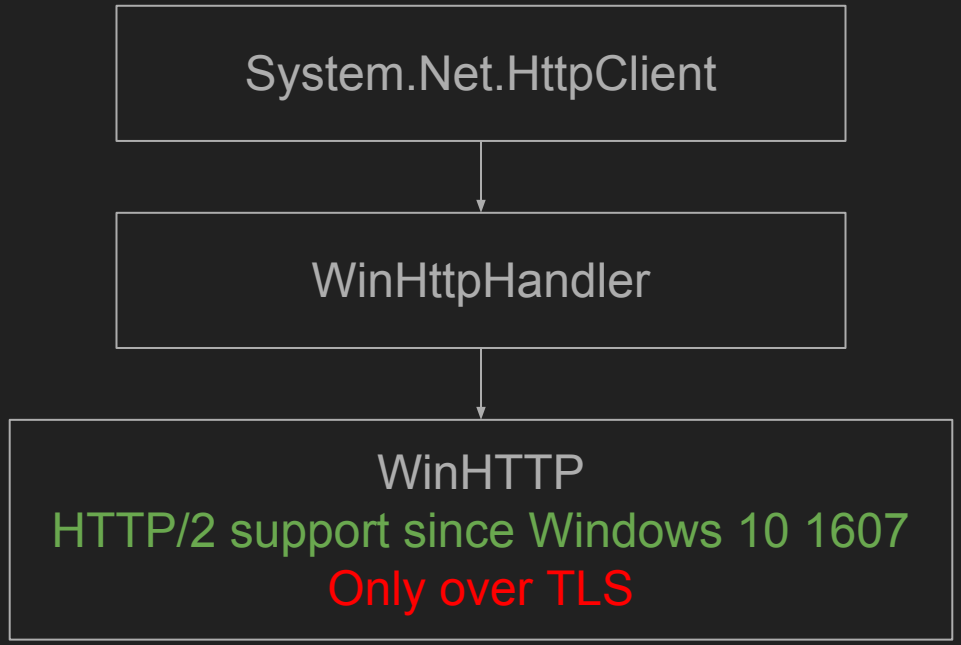
HTTP/1.1 client on .NET Framework



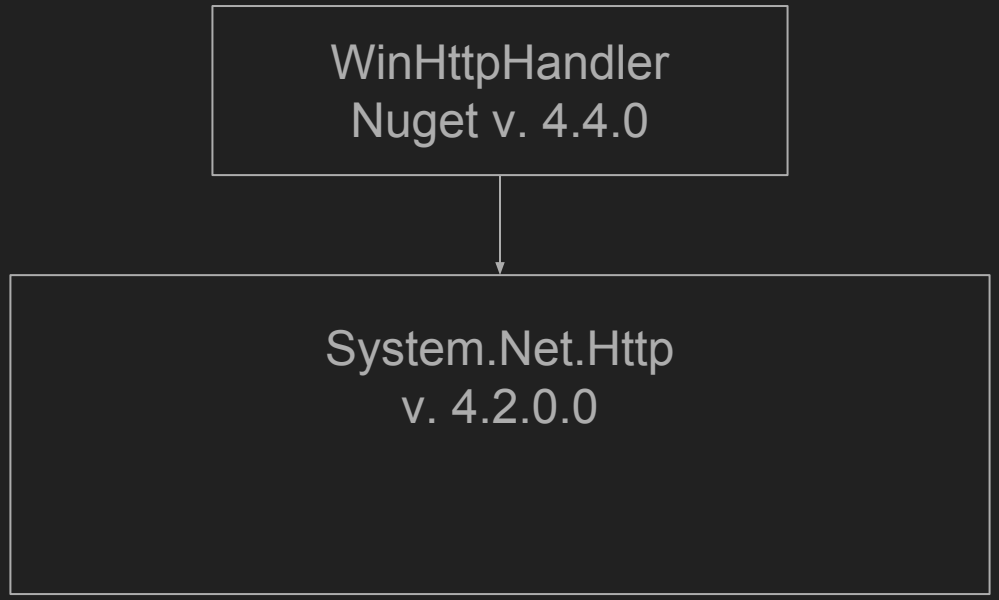
HTTP/2 server on .NET Framework



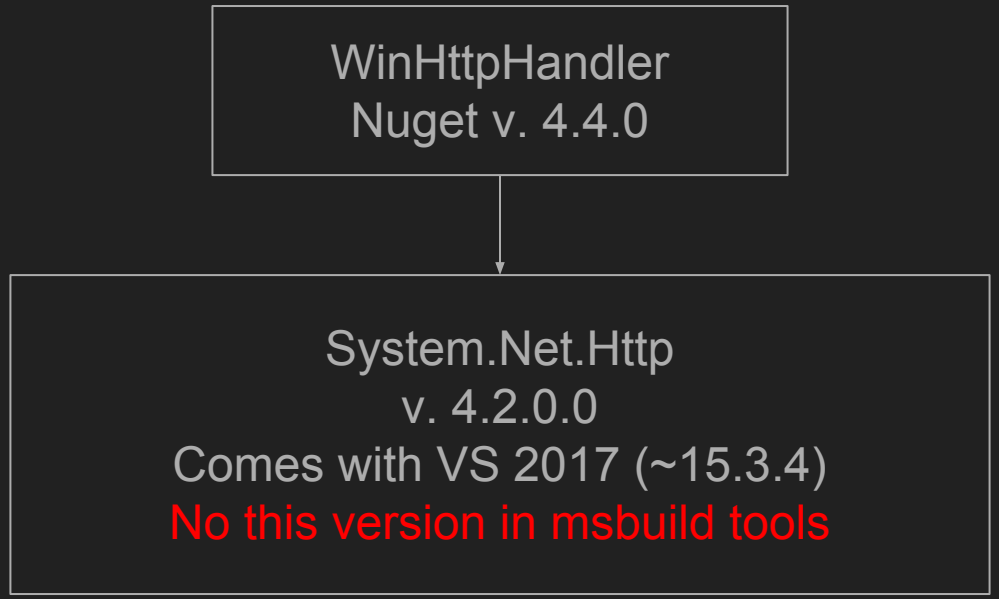
HTTP/2 client on .NET Framework



Version hell



Version hell



Finally worked

```
using (var client = new HttpClient(new WinHttpHandler(), true))
{
    var request = new HttpRequestMessage(HttpMethod.Get, "https://google.com/")
    {
        Version = new Version(2, 0)
    };
    var response = await client.SendAsync(request);
    Console.WriteLine($"{response.StatusCode} {response.Version}");
}
```

```dotnet
// client
var n = 100;
var tasks = new Task<HttpResponseMessage>[n];
for (var i = 0; i < n; i++)
{
    tasks[i] = SendAsync();
}
var result = await Task.WhenAll(tasks);

// server
while (true)
{
    var context = listener.GetContext();
    Task.Run(async () =>
    {
        await Task.Delay(2000);
        context.Response.StatusCode = 200;
        context.Response.Close();
    });
}
```

```
> netstat -a -n | findstr 4443

TCP 0.0.0.0:4443 0.0.0.0:0 LISTENING
TCP [::]:4443       [::]:0       LISTENING
TCP [::1]:4443   [::1]:53079 ESTABLISHED
TCP [::1]:53079 [::1]:4443   ESTABLISHED
```

```
// client
var n = 100;
var tasks = new Task<HttpResponseMessage>[n];
for (var i = 0; i < n; i++)
{
    tasks[i] = SendAsync();
}
var result = await Task.WhenAll(tasks);

// server
while (true)
{
    var context = listener.GetContext();
    Task.Run(async () =>
    {
        await Task.Delay(2000);
        context.Response.StatusCode = 200;
        context.Response.Close();
    });
}
```

```
> test.exe client 4443 100

Got response 1 in 00:00:02.0031573
…
Got response 99 in 00:00:02.0011053
Got response 100 in 00:00:02.016546
```

```
// client
var n = 100;
var tasks = new Task<HttpResponseMessage>[n];
for (var i = 0; i < n; i++)
{
    tasks[i] = SendAsync();
}
var result = await Task.WhenAll(tasks);

// server
while (true)
{
    var context = listener.GetContext();
    Task.Run(async () =>
    {
        await Task.Delay(2000);
        context.Response.StatusCode = 200;
        context.Response.Close();
    });
}
```

> test.exe client 4443 101

Got response 1 in 00:00:02.0031573
…
Got response 99 in 00:00:02.0011053
Got response 100 in 00:00:02.016546
Got response 101 in 00:00:04.072555

Where is the limit?

ServicePoint.ConnectionLimit

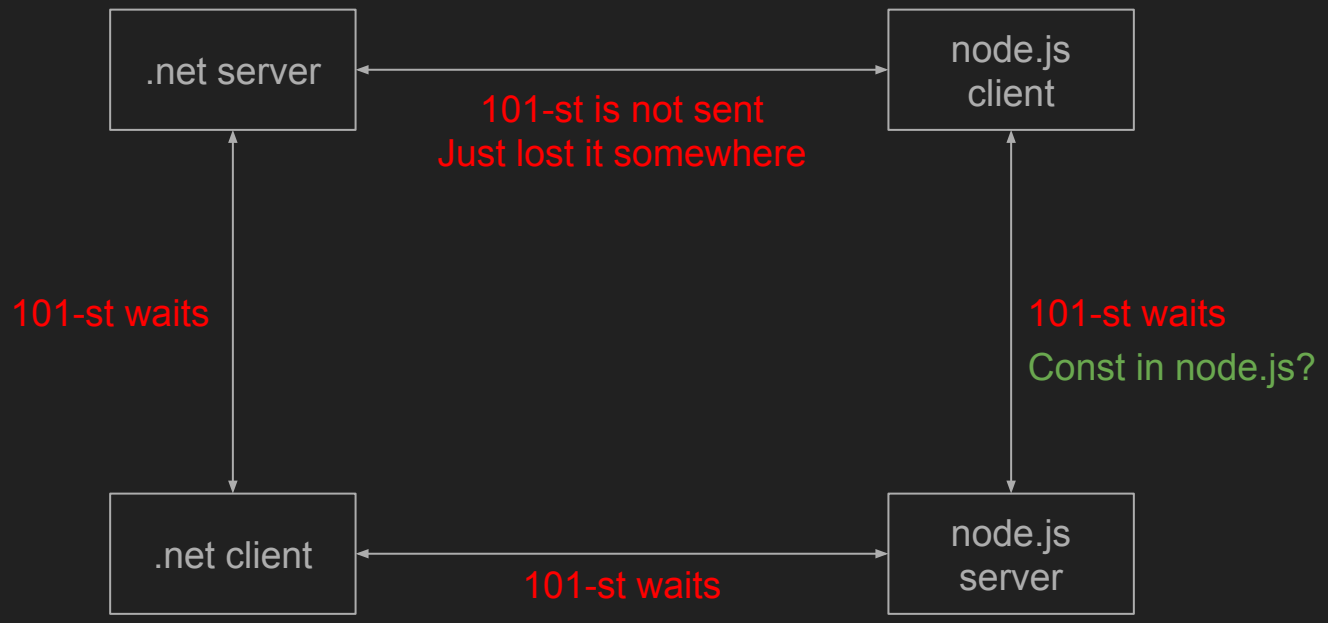
WinHttpHandler.MaxConnectionsPerServer

Const inside WinHttpHandler?

Const inside http.sys?

Let’s try another server/client

Different server/client



Mystical constant 100

https://github.com/nodejs/node/blob/master/src/node_http2.cc#L58

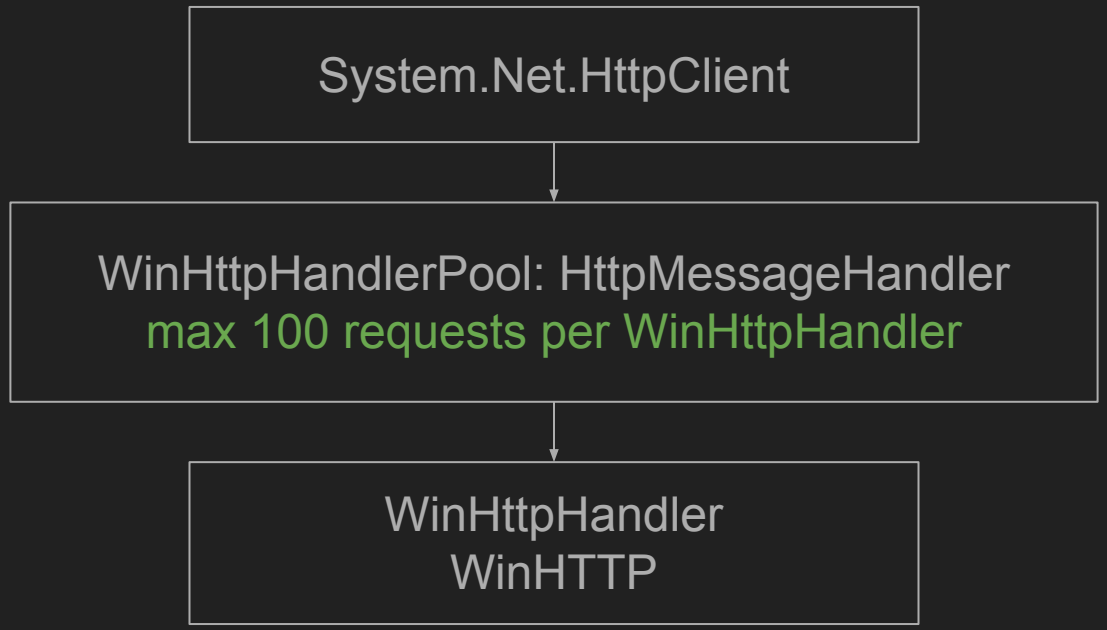
```
57    // Recommended default
58    nghttp2_option_set_peer_max_concurrent_streams(options_, 100);
59    if (flags & (1 << IDX_OPTIONS_PEER_MAX_CONCURRENT_STREAMS)) {
60      nghttp2_option_set_peer_max_concurrent_streams(
61          options_,
62          buffer[IDX_OPTIONS_PEER_MAX_CONCURRENT_STREAMS]);
63    }
```

HTTP/2 RFC 7540

SETTINGS_MAX_CONCURRENT_STREAMS (0x3): Indicates the maximum number of concurrent streams that the sender will allow. This limit is directional: it applies to the number of streams that the sender permits the receiver to create. Initially, there is no limit to this value. It is recommended that this value be no smaller than 100, so as to not unnecessarily limit parallelism.

Tuned node.js client

WinHttpHandlerPool



Load test

How many long-poll reqs can we have at once?

Theory: 65k * 100 = 6.5kk

Guess in practice?

~27k, and client thread pool exhausted

Thread pool

Async operations: promise, do smth when completes

Thread can do other work while operation is pending

Thread goes back to ThreadPool

min_threads / max_threads

First min_threads threads are created instantly

Then threads are created with delay (~1 sec) until max_threads

After max_threads no more threads are spawned

In practice when min_threads is reached nothing works

WinDbg

~*e !CLRStack

```
00000022854f8bc8 00007ff8c6d46964 [HelperMethodFrame: 00000022854f8bc8] System.Threading.Monitor.Enter(System.Object)
00000022854f8cc0 00007ff8b2388d72 System.Threading.TimerQueueTimer.Change(UInt32, UInt32)
00000022854f8d40 00007ff8b239f214 System.Threading.CancellationTokenSource.CancelAfter(Int32)
00000022854f8da0 00007ff8b239f14b System.Threading.CancellationTokenSource.CancelAfter(System.TimeSpan)
00000022854f8dd0 00007ff8545bc8d5 Kontur.Clusterclient.Extensions.SystemNetHttp.SystemNetHttpTransport+<SendAsync>d__8.MoveNext()
```

System.Threading.Monitor.Enter(System.Object)

System.Threading.TimerQueueTimer.Change(UInt32, UInt32)

System.Threading.CancellationTokenSource.CancelAfter(Int32)

Timeouts

```
var request = new HttpRequestMessage();
var timeout = TimeSpan.FromSeconds(30);
using (var cts = new CancellationTokenSource())
{
    cts.CancelAfter(timeout);
    var response = await client.SendAsync(request, cts.Token);
}
```

TimerQueue: reference source

```
// The one-and-only TimerQueue for the AppDomain.
static TimerQueue s_queue = new TimerQueue();
```

TimerQueue: reference source

```
// We use a single native timer, supplied by the VM,
// to schedule all managed timers in the AppDomain.
```

TimerQueue: reference source

```
// Note that all instance
// methods of this class
// require that the caller hold
// a lock on TimerQueue.Instance.
```

TimerQueue: reference source

```
// The data structure we've chosen
// is an unordered doubly-linked list of active timers.
// This gives O(1) insertion and removal,
// and O(N) traversal when finding expired timers.
```

Lock convoy

There are 27k CancellationTokenSources for request timeouts

CTS.CancelAfter/CTS.Dispose/etc… executes on ThreadPool

CTS uses System.Threading.Timer

TimerQueue works under lock

In long-polling, CTS are long-lived (~30 sec)

Many long-lived timers make lock convoy

Custom timeouts

Implemented CustomTimerQueue

No VM timer: just a background thread

BinaryHeap with single lock for insert

Lock sharding: there are 256 CustomTimerQueue instances

Load test

How many long-poll reqs can we have at once?

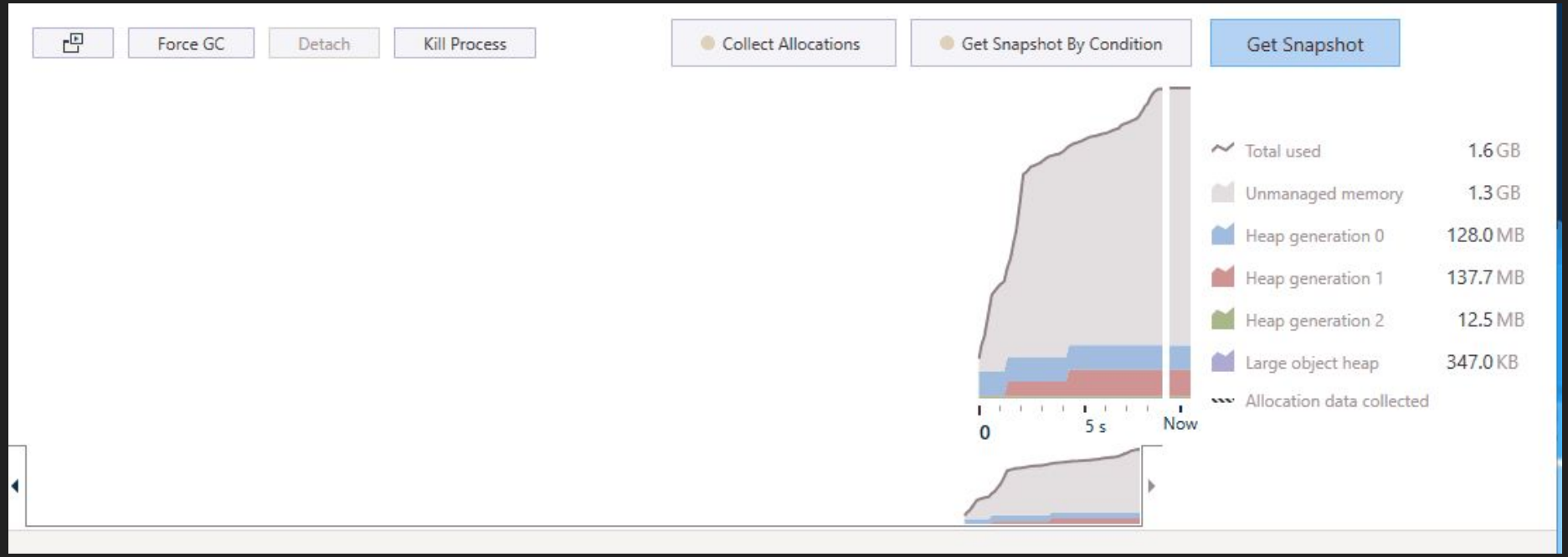
theory: 6.5kk, practice: 27k

After timeouts optimization?

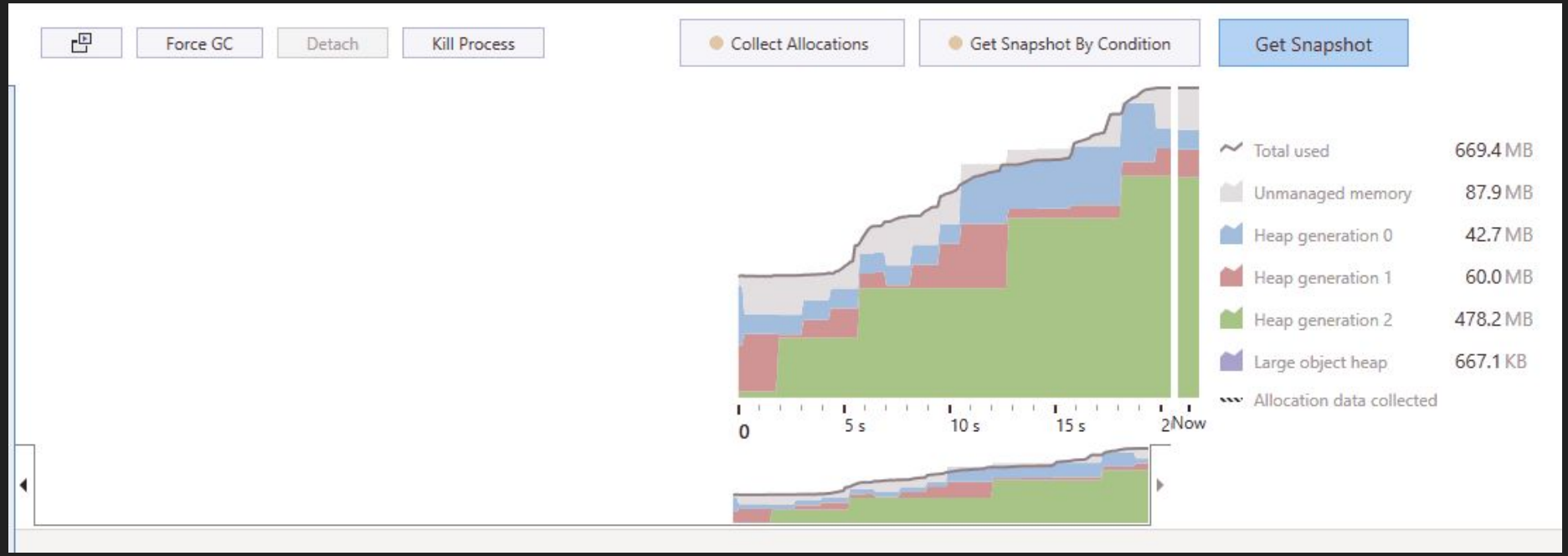
~57k and we are out of memory

Name	PID	Status	CPU	Working set (m...	Memory (private...	Commit size
LongPollTest.exe	10516	Running	60	1,984,408 K	1,956,832 K	4,669,760 K

4GB, HTTP/2


Force GC
Detach
Kill Process
Collect Allocations
Get Snapshot By Condition
Get Snapshot
Total used 1.6 GB
Unmanaged memory 1.3 GB
Heap generation 0 128.0 MB
Heap generation 1 137.7 MB
Heap generation 2 12.5 MB
Large object heap 347.0 KB
Allocation data collected
0
5 s
Now

4GB, HTTP/1.1


Force GC
Detach
Kill Process
Collect Allocations
Get Snapshot By Condition
Get Snapshot
Total used 669.4 MB
Unmanaged memory 87.9 MB
Heap generation 0 42.7 MB
Heap generation 1 60.0 MB
Heap generation 2 478.2 MB
Large object heap 667.1 KB
Allocation data collected
0
5 s
10 s
15 s
Now

The end..?

WinHttpHandler doesn’t meet our expectations

Strange memory allocations

Little customization for HTTP/2

Http.sys reacts strangely on SETTINGS frame

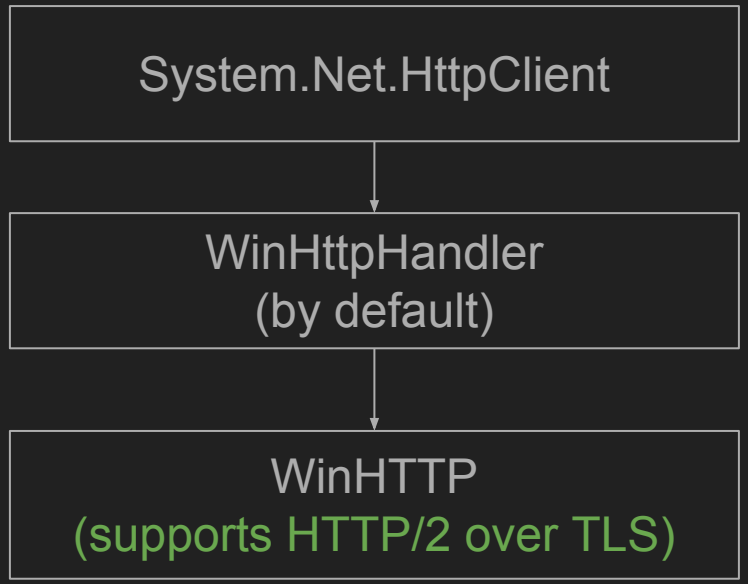
Haven’t investigated thoroughly

What if… .NET Core on Windows?

Server

What if… .NET Core on Windows?

Client



What if… .NET Core?

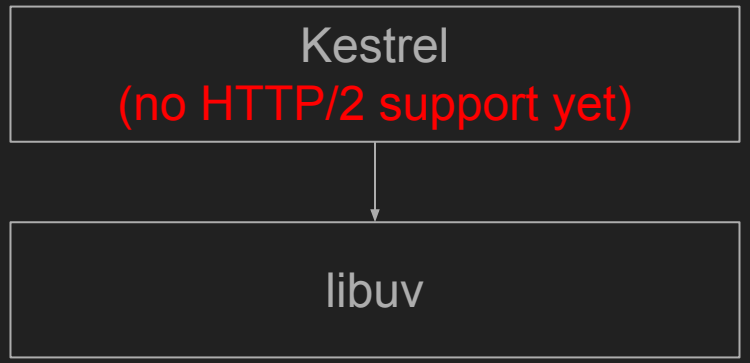
TimerQueueTimer

Lock convoy issue: https://github.com/dotnet/coreclr/issues/13083

Fixed by lock sharding: https://github.com/dotnet/coreclr/pull/14527

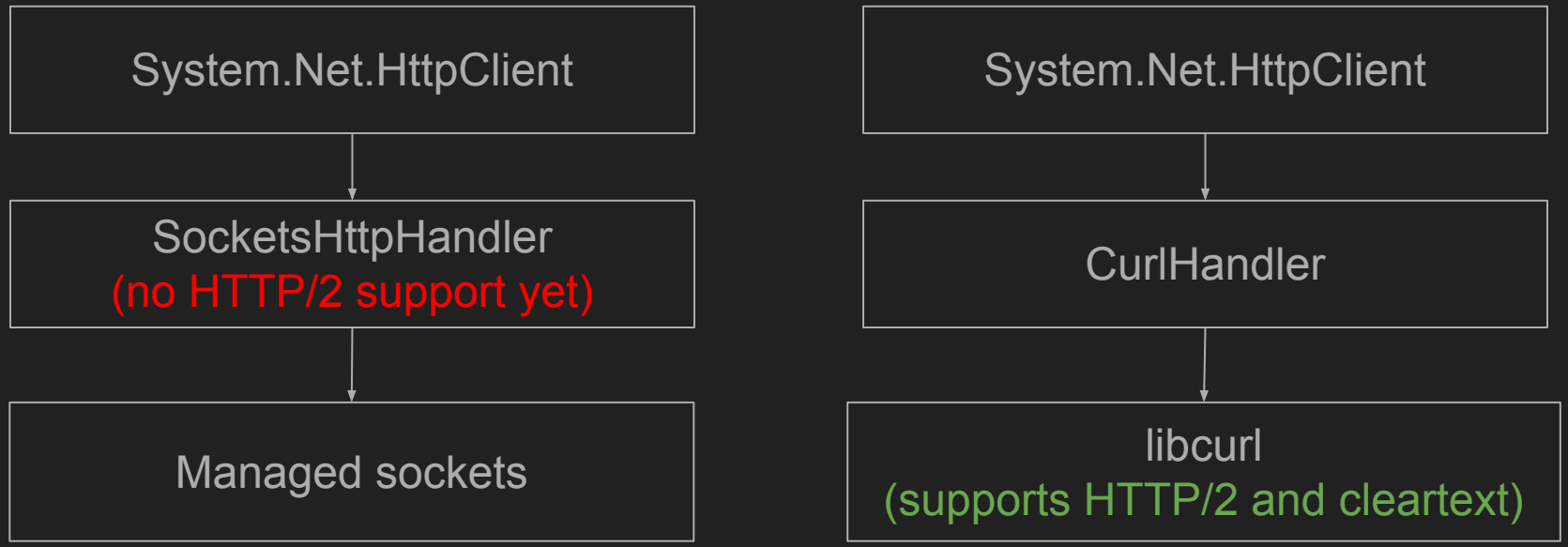
What if… .NET Core on Linux?

Server



What if… .NET Core on Linux?

Client



```
// client
var n = 100;
var tasks = new Task<HttpResponseMessage>[n];
for (var i = 0; i < n; i++)
{
    tasks[i] = SendAsync();
}
var result = await Task.WhenAll(tasks);

// server
while (true)
{
    var context = listener.GetContext();
    Task.Run(async () =>
    {
        await Task.Delay(2000);
        context.Response.StatusCode = 200;
        context.Response.Close();
    });
}
```

Multiplexing doesn't work
(out of the box)
Ubuntu 16.04
Dotnet 2.1.101 (2.0.6)

> test.exe client 4443 100

Got response 1 in 00:00:02.0011053
Got response 2 in 00:00:04.0165462
Got response 3 in 00:00:06.2270573
...

Old libcurl?

HTTP/2 since 7.33.0

Supports multiplexing since 7.43.0

We have… 7.42.0

Updated to 7.59.0…

Nothing changed

CurlHandler

```
static CurlHandler()
{
    Interop.Http.CurlFeatures features = Interop.Http.GetSupportedFeatures();
    //some code...
    s_supportsHttp2Multiplexing =
        (features & Interop.Http.CurlFeatures.CURL_VERSION_HTTP2) != 0
        && Interop.Http.GetSupportsHttp2Multiplexing();
```

s_supportsHttp2Multiplexing == false

CurlHandler

```
[DllImport(Libraries.HttpNative, EntryPoint = "HttpNative_GetSupportsHttp2Multiplexing")]
– references | stephentoub, 757 days ago | 1 author, 1 change
internal static extern bool GetSupportsHttp2Multiplexing();
```

CurlHandler

```
#define MIN_VERSION_WITH_CURLPIPE_MULTIPLEX 0x074300

extern "C" int32_t HttpNative_GetSupportsHttp2Multiplexing()
{
    curl_version_info_data* info = curl_version_info(CURLVERSION_NOW);
    return info != nullptr &&
        (info->version_num >= MIN_VERSION_WITH_CURLPIPE_MULTIPLEX) &&
        ((info->features & PAL_CURL_VERSION_HTTP2) == PAL_CURL_VERSION_HTTP2) ? 1 : 0;
}
```

This means 7.67.0!
Should be 0x072b00

Bug in hex

Latest libcurl: 7.60.0 → 0x073C00

Check in code 0x074300 → 7.67.0

https://github.com/dotnet/corefx/pull/29213

Let’s hack libcurl: 7.67.0-DEV

```
// client
var n = 100;
var tasks = new Task<HttpResponseMessage>[n];
for (var i = 0; i < n; i++)
{
    tasks[i] = SendAsync();
}
var result = await Task.WhenAll(tasks);

// server
while (true)
{
    var context = listener.GetContext();
    Task.Run(async () =>
    {
        await Task.Delay(2000);
        context.Response.StatusCode = 200;
        context.Response.Close();
    });
}
```

Ubuntu 16.04
Dotnet 2.1.101 (2.0.6)
libcurl 7.67.0-DEV

```
> test.exe client 4443 100
…
Got response 99 in 00:00:02.0017
Got response 100 in 00:00:02.04465
Got response 101 in 00:00:04.2270
...
```

What’s next?

HTTP/2 support in SocketsHttpHandler (future)

HTTP/2 support in Kestrel (work in progress)

Cleartext support (?)

Links

http://bit.ly/dotnext-2018-http2 - slides

https://github.com/ezsilmar/dotnext-2018-http2 - code from this talk

https://t.me/windbg - kontur windbg chat and cheatsheet (russian)

https://tools.ietf.org/html/rfc7540 - HTTP/2 RFC, a reading for the rainy day

ezsilmar@gmail.com, https://t.me/ezsilmar - contact me

CurlHandler.SendAsync

```
// In support of HTTP/2, enable HTTP/2 connections to be multiplexed if possible.
// We must only do this if the version of libcurl being used supports HTTP/2 multiplexing.
// Due to a change in a libcurl signature, if we try to make this call on an older libcurl,
// we'll end up accidentally and unconditionally enabling HTTP 1.1 pipelining.
if (s_supportsHttp2Multiplexing)
{
    ThrowIfCURLMError(Interop.Http.MultiSetOptionLong(multiHandle,
                Interop.Http.CURLMoption.CURLMOPT_PIPELINING,
                (long)Interop.Http.CurlPipe.CURLPIPE_MULTIPLEX));
    EventSourceTrace("Set multiplexing on multi handle");
}
```