Connection Management is HARD

HTTP Workshop 2017, London

It used to be “simple”

Need to make request, ~~open a new connection~~

if an idle connection is open, use that

else, make a new connection (unless there are too many already)

h2: Multiplexing

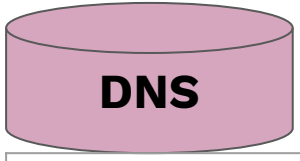
An h2 connection pretty much always counts as idle

The stream concurrency limit looks like the cap on connections to the same origin

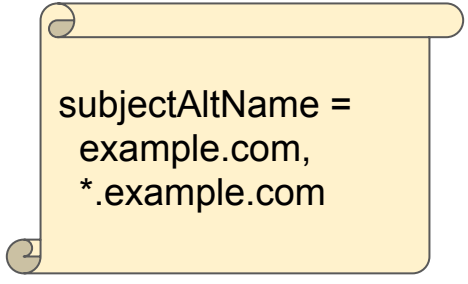
Not really a big change

h2: Coalescing

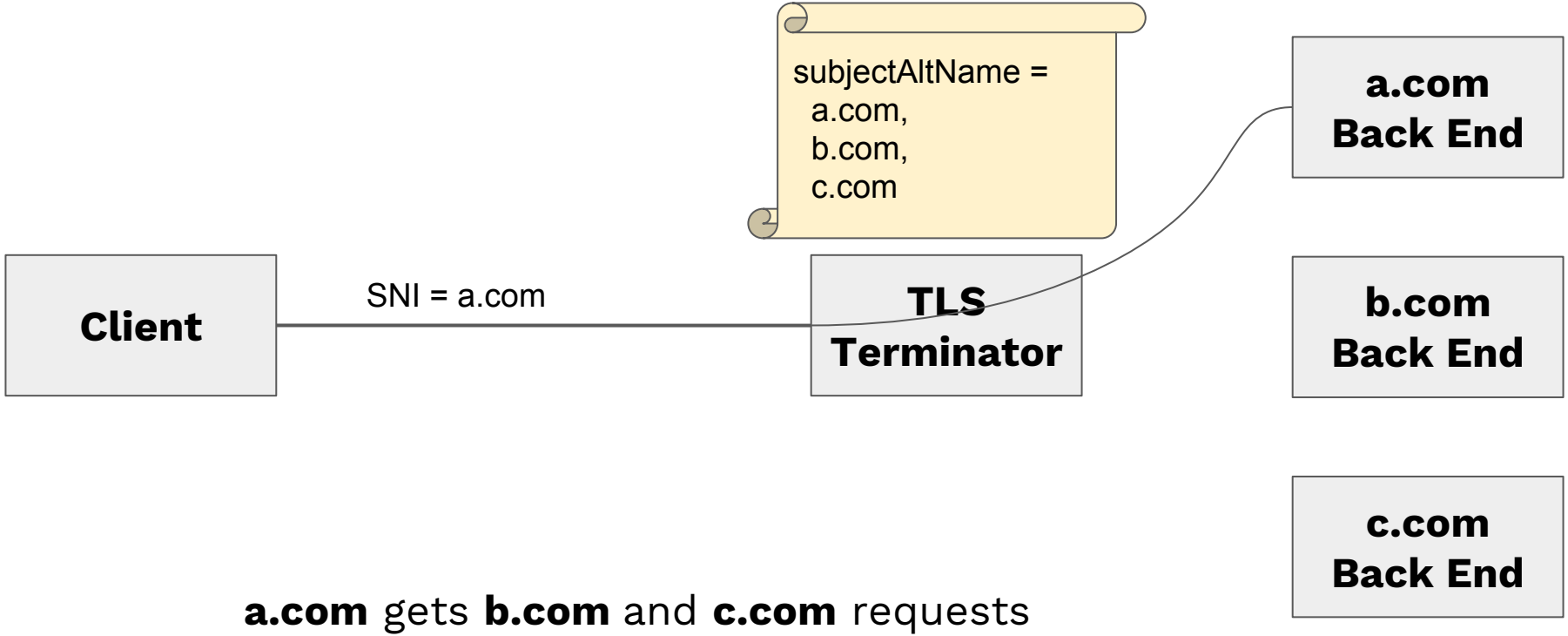
When you need a new connection and the IP address turns out to be the same* as an existing connection, then check the certificate



example.com	192.0.2.75	192.0.2.78	2001:0db8::f2	2001:0db8::f4
www.example.com	192.0.2.75		2001:0db8::f2	2001:0db8::f4



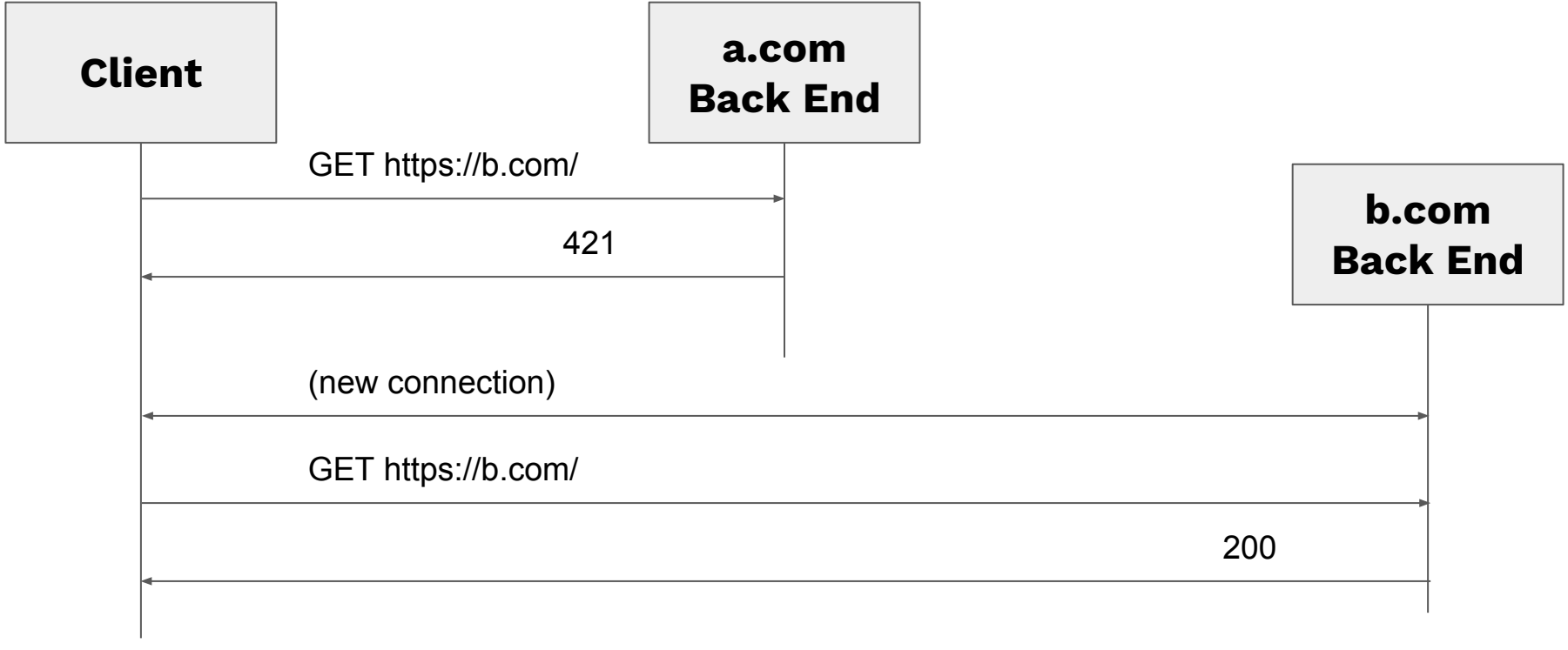
Coalescing -> Operational Challenges



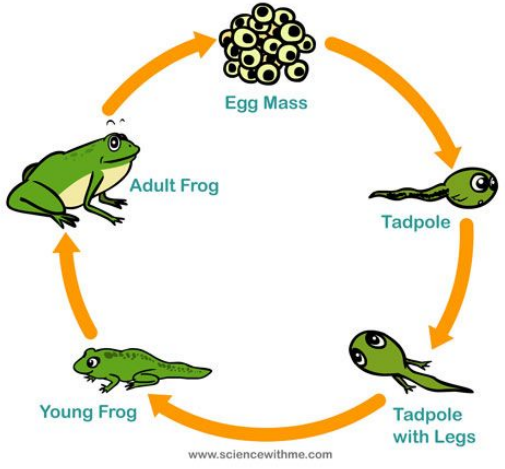
a.com gets **b.com** and **c.com** requests

421 (Not Authoritative)

We added 421 to deal with this problem, but it's a little slow



ORIGIN Frame



A turgid history of changes

- First it was simple: a list of origins
- It grew statement management: additions, removals, flushes
- A minimalist “nothing other than the SNI” option was considered
- But ended up closer to the initial design

ORIGIN Frame Today

Initially a connection is valid for any host in the certificate (h2 default)

An ORIGIN frame indicates support for the feature

The set of origins is then

- a single origin (https, SNI hostname, server port**)
- plus origins that are included in ORIGIN frames
- less origins that generate responses with a 421 status code

Alt-Svc

Used to prime cross-protocol upgrades (opp-sec, QUIC)

Alt-Svc: h2=”x.example.net:443”
= Connect to x.example.net instead

Alt-Svc: h2=”:8443”
= Connect to port 8443 instead

Alt-Svc: hq=”:443”
= Use QUIC

Alt-Svc

CNAME at the HTTP layer

- Doesn’t affect authority
- Changes ALPN protocol, DNS lookup, and (TCP) port

Make before break

- An invisible transition is great because users don’t notice
- which is **terrible** because it’s hard to see when the alternative breaks

Secondary Certificates

h2 allows coalescing, but everything has to fit in one certificate

For example: *.www.yahoo.com, add.my.yahoo.com, *.att.yahoo.com, att.yahoo.com, au.yahoo.com, be.yahoo.com, brb.yahoo.com, br.yahoo.com, ca.my.yahoo.com, ca.rogers.yahoo.com, ca.yahoo.com, ddl.fp.yahoo.com, de.yahoo.com, en-maktoob.yahoo.com, espanol.yahoo.com, es.yahoo.com, fr-be.yahoo.com, fr-ca.rogers.yahoo.com, frontier.yahoo.com, fr.yahoo.com, gr.yahoo.com, hk.yahoo.com, hsrd.yahoo.com, ideanetsetter.yahoo.com, id.yahoo.com, ie.yahoo.com, in.yahoo.com, it.yahoo.com, maktoob.yahoo.com, malaysia.yahoo.com, my.yahoo.com, nz.yahoo.com, ph.yahoo.com, qc.yahoo.com, ro.yahoo.com, se.yahoo.com, sg.yahoo.com, tw.yahoo.com, uk.yahoo.com, us.yahoo.com, verizon.yahoo.com, vn.yahoo.com, www.yahoo.com, yahoo.com, za.yahoo.com, zed.yahoo.com

Idea: Add a Certificate to a Connection

This would allow a server to have many small certificates

- certificates can be added as needed

Great for servers that have multiple names

Especially good for servers with multiple disparate names

This could also fix or help with

- post-handshake client authentication
- domain fronting for privacy reasons

How it works

TLS exported authenticators copy the authentication messages from TLS into a blob that can be used in a higher layer protocol

On Stream 0:

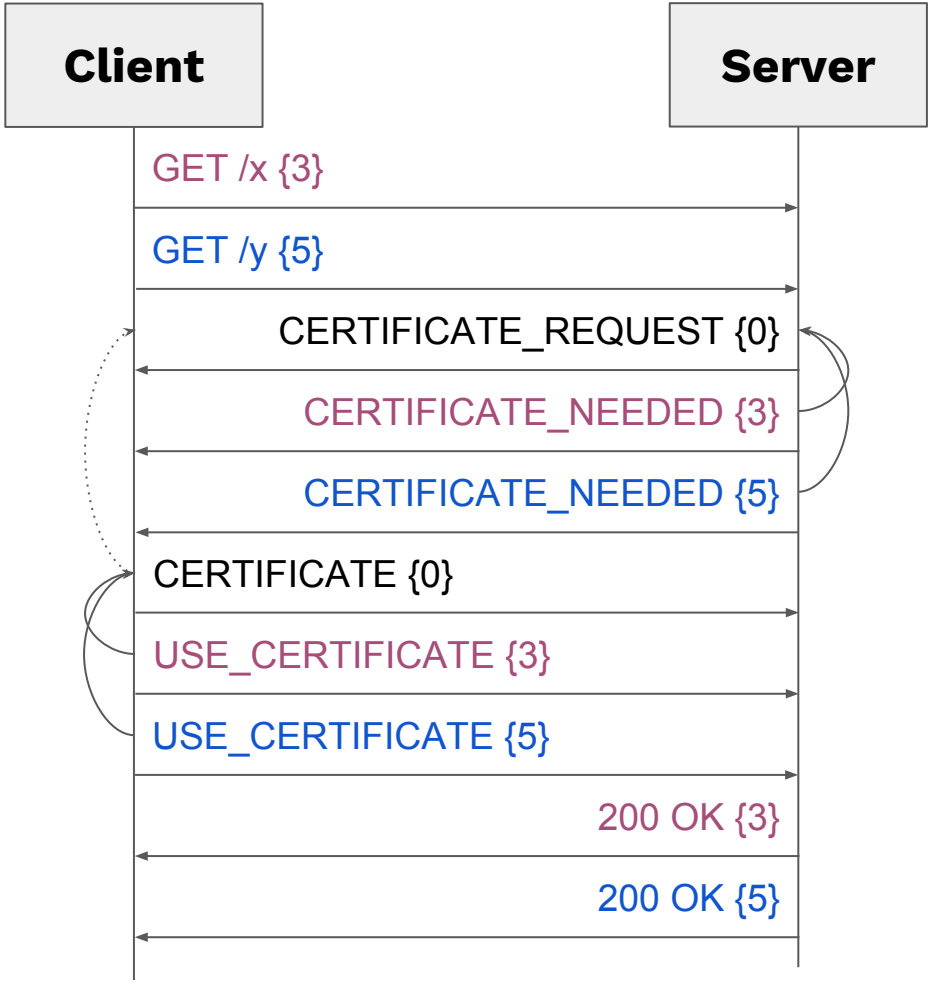
- An endpoint can send an authenticator with CERTIFICATE
- An endpoint can request a CERTIFICATE using CERTIFICATE_REQUEST

After make or receiving a request:

- An endpoint can reference a certificate with USE_CERTIFICATE
- Request USE_CERTIFICATE by sending CERTIFICATE_NEEDED

Examples


Client
Server
CERTIFICATE (x.com)
ORIGIN (https://x.com)


Client
Server
GET /x {3}
GET /y {5}
CERTIFICATE_REQUEST {0}
CERTIFICATE_NEEDED {3}
CERTIFICATE_NEEDED {5}
CERTIFICATE {0}
USE_CERTIFICATE {3}
USE_CERTIFICATE {5}
200 OK {3}
200 OK {5}