Known startup state for a simpler and more robust HTTP 2.0

HTTPbis WG, IETF 85, Orlando
15 March, 2013

Osama Mazahir
Matthew Cox
Gabriel Montenegro
(Microsoft)
The issue: unknown startup state

• Needless complexity if the protocol does not start at a known state at both client/server
• Best to not allow the protocol to “overstep” itself
  – “overstep”: send more than you have credit for, open more streams than the receiver allows for, etc.
• Let’s not abandon protocol correctness in the quest for speed (besides, no need to)
• Can lead to more overstepping with future extensions with unpredicable consequences
• We can solve this
Some related issues

#51: Client advertising settings during Upgrade dance
   – Client sends GET with Upgrade
   – Server responds with 101 followed by SYN_REPLY, DATA, maybe PUSHed streams
   – Could blow up client’s buffer’s or PUSH against client wishes
   – New HTTP header?

#38: SETTINGS_MAX_CONCURRENT_STREAMS
   – TLS case: Client opens too many streams
     • TLS extra info?
   – Upgrade case: Server opens too many streams
     • New HTTP header?

#40: Defaulting to no-push via SETTINGS_MAX_CONCURRENT_STREAMS
   – Mailing list discussion: explicit signal is better
   – New HTTP header?

#?? (new issue): Unknown window credits
   – TLS case: Client sends too much to the server
     • TLS extra info?
   – Upgrade case: Server sends too much to the client
     • New HTTP header?
Current alternatives and their issues

- **SETTINGS frame? Too late**
  - Client could send its initial settings frame and simultaneously open too many streams or send too much
  - In Upgrade scenario, Server could respond to HTTP/1.1 GET with 101 followed by PUSH with too many streams too much data
- **wait for the other’s SETTINGS frame?**
  - Time wasted
- **Defaults? Wishful thinking...**
  - Hard to agree, discussion can rathole due to different scenarios
  - Does not allow startup state to vary from defaults
- **DNS? Not always possible**
  - Good plan B, though.
Proposal: set startup state in negotiation

- During negotiation allow one party to inform the other of its desired startup state

- Potential state to set (choose which, if any, to communicate)
  1. Max concurrent streams and Receive window
  2. “Neither PUSH nor inlining desired”

- Potential Solutions (choose which, if any, to pursue):
  1. In Upgrade case:
     • Client sends HTTP header(s) to set desired state
  2. In TLS case
     • Enhance TLS protocol negotiation by allowing sending the desired startup state
     • Reuse “TLS Handshake Message for Supplemental Data” (RFC4680)?
EXTRA SLIDES
client ignores server max streams

• client opens too many streams to the server (client need not wait for SETTINGS)
• Server resets all streams over its limit
• Client must keep canceled streams in “backorder”, issuing them as outstanding requests are served
• Client must buffer canceled request, send an error to the application, etc, leading to complexity and app issues
• If max streams is known, any server cancel can be dealt with simply (shutting down the connection)
server ignores client max streams

• In Upgrade scenario, client sends http/1.1 GET, server sends 101, immediately followed by HTTP/2.0 traffic
• Can result in server PUSHing too many streams, before the client has a chance to send a SETTINGS frame
• Wasteful of resources at both sides (e.g., data allowance, battery on the client)
• Complexity at the server: push streams in backorder, issued gradually
Unknown Window Credits (2)

• Client or server (in Upgrade case) send too much
• Must keep count of “negative” credits, increasing code complexity
• Worse: hard to reason about protocol correctness
• Sane Transport Protocols DO NOT DO THIS (TCP, etc)
• Will Flow Control Algorithms now have to take into account handling of “negative” credit?
  – Raises the bar for flow control algorithms
  – Defeats the purpose of flow control principles
• We are creating a transport protocol: we should follow best practices